

The Industrial Use of a Formal Method in a Gas Turbine Engine Electronic Control System

Extended Abstract

J.R. Blow, A.J. Galloway, J.A. McDermid,
High Integrity Systems Engineering,
Department of Computer Science,
University of York, UK

M.G. Dowding, T.J. Cockram,
Rolls-Royce plc
Bristol, UK

February 29, 2000

Introduction

Defence Standard 00-55 [1] requires the use of formal methods in the specification, and subsequent analysis, of safety critical control software. To date, however, there has been reluctance from industry to adopt these methods. This has been due in part to the high costs of producing a formal specification compared with conventional specifications and the need for specially trained mathematicians to write and understand the formal software specification. In addition, formal specifications have generally only been used for small-scale projects as academic exercises, such as library systems, or in a limited way, for the most critical systems, such as nuclear plant protection. Examples of the successful use of formal methods within real time industrial control systems are, therefore, very limited. For example, the SHOLIS Technical Demonstrator [2]¹.

In an attempt to redress this, and in doing so make formal methods more accessible to control systems engineers, the Practical Formal Specification (PFS) technique was developed. Funded by the UK Ministry of Defence, it aims to provide the technology for formal development of engine control software. The work is supported by the Rolls-Royce Aeroengines group who, as a supplier to the Ministry of Defence, can be required to use (or mandate of their subcontractors) formal techniques for the safety critical components of the software they provide. The PFS project also focuses on flight control avionics systems, and is additionally supported by BAe Systems.

Domain Specific Formal Notations for Engine Control Software

Although some departments of Rolls-Royce have investigated the use of discrete formal methods in the past, such techniques are still not, in general, used in the development process for engine control software. Indeed, their use is still very much an area for research and development.

To understand the reasons why formal techniques are not in more widespread use it is important to appreciate the domain in terms of both the process and the product it produces. Control software is developed within a multi-level concurrent engineering process [3] where airframe requirements give rise to propulsion system level requirements, which in turn give rise to control system requirements. The control system itself is made up of many engineered components, including sensors, actuators, hydraulics, digital hardware and software. Software requirements hail from, and are affected by, the requirements and design of all the components in the engineering hierarchy.

¹ Although, unlike engine control systems, which are 'Full-authority', the SHOLIS system was an advisory system, with full control remaining with a human being (a pilot).

Obstacles to the Use of Formal Development

The engineering process needs to be foremost in mind. However, there are also problems with formal specification and verification even if one takes a static view of the software requirements and their subsequent design. Our experience with the PFS study has highlighted three main obstacles to the adoption of the kind of formal development advocated within Defence Standard 00-55. These are the inapplicability of formal abstraction mechanisms, the difficulty of eliciting and validating formal specifications, and the narrow viewpoint of the established formalisms.

Domain Specific Language

The PFS approach [4,5] to solving these problems has been to construct a Domain Specific Language. By Domain Specific Language (DSL) we mean a language with a syntax ideally tailored to expressing the basic concepts and structuring mechanisms of a particular application area. Thus, rather than being limited only by the bounds of what one can describe in, say, set theory, the practitioner is limited by the kinds of requirements concepts and structures of control software. The most important features of a DSL are an intuitive concrete representation, a layered formal semantic construction and a set of reasoning goals.

In PFS, the domain specific language is based heavily on hierarchical state machines (such as Statecharts [6]) and tabular notations such as those employed in SCR ('Software Cost Reduction') e.g. [7]. The DSL organises requirements as state-based components (which maintain behavioural or information state), reactive components (which have no state) and aggregations thereof. The formal semantics is inspired by work on the formalisation of graphical notations and enables the formulation of proof obligations. Proof obligations may then be generated for model "healthiness" thus increasing confidence in the validity of a specification. Additional reasoning obligations may be generated in the light of a design artifact to demonstrate compliance to a specification given appropriate refinement relationships. Pragmatically, reasoning goals can then be handed to experts in formal reasoning.

Domain Specific Abstraction

Domain specific notations permit domain specific abstraction. By constraining the domain of application it is possible to be more specific about what forms of abstraction are applicable. Given the relative importance or criticality of the functional components that make up an application in a particular domain, a specifier can employ structural abstraction to manage the introduction of detail. For example, in PFS the emphasis is on specifying core control requirements and working 'outwards', through signal validation and failure management, toward the sensor and actuator interfaces.

Additional abstractions can be provided within the notation itself. In PFS the use of hierarchical state machines permits the specifier to intuitively describe the desired behaviour, rather than stating behaviour in terms of the control variables that will eventually be used in the implementation. Also, the PFS approach focuses on the discrete aspects of engine control requirements. The notation prompts the specifier to provide 'place holder' information for the transformations being developed using continuous mathematics. Such requirements will be validated independently by their own mathematically 'formal' method.

A major consideration is how different levels of abstraction can be related formally. Again this can be domain specific, although PFS relies heavily on the abstraction mechanism of stating assumptions and specifying within those assumptions. The emphasis is on explicitly stating the assumptions that a discrete requirement (or placeholder for a 'continuous' requirement) expects in its context. Assumptions might concern the range of inputs to a requirement, the rate of change or the expected relationship between inputs (especially with respect to arithmetic saturation etc.).

Composition

The process of validating a set of requirements is formally supported by composing requirements together to form aggregates and propagating assumptions outward to the environment. Formal composition is based on the concepts of guarding and governing of requirements. Of these governing is the most straightforward, analogous to the concept of weakest precondition derivation in programming language semantics e.g. generalised substitution language [8]. Guarding takes into account requirements independence. That is, the conditions under which a requirement will not affect the observable behaviour of the software of which it is a part. Composition of requirements amounts to deriving the weakest precondition over a set of inputs to guarantee that, where a requirement affects the observable behaviour of the software, its assumptions are guaranteed by its context.

In general, given requirement composition in terms of guarding and governing the usual 'weakening of the precondition' refinement relation holds. The relationship supports 'ideal' to 'real' refinement in PFS. When an engineer initially describes a requirement, they assume an ideal context, e.g. no noise on signals, no failures, hardware responds as a quickly as software, signals will not induce arithmetic saturation. Such simplifications result in strong assumptions

being documented for each requirement - assumptions that could not usually be guaranteed by the environment. The engineer subsequently has several 'refinement' courses. The assumptions can be systematically weakened, and behaviour added to deal with the additional situations. Alternatively, additional behaviour may be introduced to guard and govern the requirement, and thus guarantee its assumptions.

The Specification

The PFS project has made steady progress. After spending a considerable amount of time understanding the domain, a hypothesis was developed which iteratively evolved into the strategy outlined above. This is the first time that the requirements notation has been used on a large scale case study. Previously, the notation has been applied to a small case study based on the engine starting requirements of a helicopter engine controller, while the use of the notation with formal analysis was the subject of a case study involving flight control requirements.

The PFS specification is part of a Technical Demonstrator Programme for a gas turbine engine controller and is based upon the software requirements taken originally from a control law modeling tool. The model was generated by a controls engineer and the requirements simulated against an engine model. Thus, the original requirements are mathematically-based and have undergone informal validation by the controls engineer. This implies that the original tool-based specification is of a higher quality than would be expected of an informal natural language specification. The informal software specification is a document based representation of the tool-based model. Use of the tool has, perhaps, made coding easier, but the constructs available often hide the true intent. This makes it difficult for the programmer to easily identify assumptions on the use of parts of the model, which in turn could lead to the use of a requirement or portion of the model out of context.

So far, we have specified the core functionality of the engine controller, and have reasoned formally by discussion. The specification has undergone peer review about healthiness, but formal proof has yet to be undertaken.

Achievements

Generating the formal specification identified weaknesses within the existing system specification. In particular there were areas of the system specification which were open to incorrect interpretation and others where required information was missing. We feel that the discipline of recording assumptions over which the particular part of the specification operates gives a clear understanding of the design intent.

Other notable benefits have been gained from the production of the PFS specification. It has made the engineers think much more about the system under development, resulting in a greater understanding of the system. Importantly, it has encouraged the engineer to think formally about the system, *but without the need to learn an unfamiliar mathematical notation*. It has allowed them to document the process using an intuitive, rather than 'formal', notation.

The formal specification produced can be used as a medium to improve the precision and clarity of the requirements. This aids the understanding of the intended behaviour of the system, so that formal proofs can be conducted at a later stage. The specification document also allows both the engineers and the academics to understand the requirements and to use a common language. The PFS notation can therefore claim to have gone some way to overcoming the perceived problems associated with eliciting and validating requirements in formal notations. The engineers can now communicate their ideas within an intuitively appealing language.

Established formal specification techniques and methods for software are often generated separately from the system specification, and by software engineers with little appreciation of the wider system. The specification developed provides a link between systems engineering methods and specification and software specification in a formal sense, which removes the potential for discontinuity which exists in current methods.

The specification has provided a foundation for the formal verification of safety properties of the system, which were identified by HAZOP techniques described in Defence Standard 00-58 [9]. It has also provided a practical demonstration in the spirit of Defence Standard 00-55 and provides a link between the techniques required by Defence Standard 00-56 [10] to determine system safety hazards and their mitigation in the system and software design.

In doing this exercise, the intent was not necessarily to find aspects of the specification incorrect. However, it is worth noting that the errors that were found were inexpensive to correct at this stage in the development. A conventional approach may have found these errors during software testing or rig testing, at which point it would have been expensive to correct them. Some engineers have expressed their doubt that testing would have found the errors

identified by PFS at all. We have increased our confidence in the specification in general and have seen that components of the system are being used in context and under the correct assumptions. We feel that we have gone some way to show that benefits are to be had from formally specifying complex systems without the immediate need for formal proof. The process of specifying the system has been a valuable exercise, too, since we have been able to check our progress against the PFS process model. Feedback from the engineers has been instrumental in the refinement and extension of the notation. The specification has identified and confirmed possible theoretical extensions.

Future Work

The formal specification cannot be said to be complete as only the core functions of the required system have been specified. These core functions also need further refinement before they can be implemented in software code. The next stage of the industrial case study will be to generate and discharge healthiness conditions (proof obligations) for the specification. Tools have already been developed to support component level healthiness, including discharging of assumptions, completeness, consistency and new theoretical work has been completed with the aim of developing a theory of composition. We are currently in the process of implementing this theory to automatically generate proof obligations. In the short term, further work is also required to integrate the work from the formal specification with the safety case for the software such that its contribution to the overall development process is clear.

Issues concerning Defence Standard 00-55 also need addressing. The full application of the standard as written is not possible in an industrial setting and, in our opinion, is not desirable. The standard mandates or implies the use of certain formal techniques that we have found are not practical or useful in the generation of this specification. We have found, however, that it is possible to maintain sufficient mathematical rigour through the use of the PFS method to ensure that safety properties can be maintained from specification to implementation. Changes to Defence Standard 00-55 are recommended particularly in the area of the formal specification and with subsequent refinement of requirements. We feel this would provide the same intent but more practical approach.

Work is also underway to extend the notation and formal underpinning to support the specification of requirements for (asynchronous) distributed systems. The technology produced is intended to be applicable at two levels. It will permit the specification and validation of distributed control systems (which are becoming more and more attractive e.g. for weight, fault tolerance). Also, it will permit the validation of concurrently integrated systems (such as integrated modular avionics solutions to the problem of controlling astable flight) whose components (or interfaces) have been specified using a PFS-style approach.

References

- [1] 'The Procurement of Safety Critical Software in Defence Equipment', Defense Standard 00-55. UK Ministry of Defence. 1997
- [2] 'The Value of Verification: Positive Experience of Industrial Proof', S. King et al. Volume 1709 of Lecture Notes In Computer Science. 1999
- [3] 'The ConCERT Approach to Requirements Specification - Version 2', A.J. Vickers, University of York Rolls-Royce UTC Technical Report No. YUTC/TR/96.1. 1996.
- [4] 'Experiences with the Application of Discrete Formal Methods to the Development of Engine Control Software', A.J. Galloway, T.J. Cockram and J.A. McDermid, Distributed Computer Control Systems. 1998
- [5] 'Towards Industrially Applicable Formal Methods: Three Small Steps, and One Giant Leap', J.A. McDermid et al, International Conference on Formal Engineering Methods. 1998
- [6] 'Statecharts: A Visual Formalism for Complex Systems', D. Harel, Science of Computer Programming, vol. 8, pp231-274. 1987
- [7] 'Documentation of Requirements for Computer Science', A.J. van Shouwen, D.L. Parnas and J. Madey. IEEE International Symposium on Requirements Engineering. 1993
- [8] 'The B Book: Assigning Programs to Meaning', J-R. Abrial, Cambridge University Press. 1996
- [9] 'HAZOP Studies on Systems Containing Programmable Electronics', Interim Defence Standard 00-58, UK Ministry of Defence. 1996
- [10] 'Safety Management Requirements for Defence Systems', Interim Defense Standard 00-56. UK Ministry of Defence. 1991