DEVELOPING AND ASSESSING SAFETY CRITICAL SYSTEMS WITH FORMAL METHODS: THE SAFEFM PROJECT *

Victoria Stavridou †
Andrew Boothroyd ‡
Timothy Boyce §
Peter Bradley‡
Jonathan Draper§
Bruno Dutertre†
Robert Smith ¶

Abstract

The SafeFM project is pursuing research on the practical application of formal methods in the development and assessment of safety critical systems. The project has been active for almost two years and it is the intent of this paper to report on our interim results. We review the aims and background of the project and report on work involving a mix of formal methods technology and existing best practice. We present a methodology for constructing coherent safety critical system specifications and critically evaluate its application to an avionics case study. We also describe our efforts in producing a practical combination of formal and structured techniques, initial work in the utilisation of formal methods for risk assessment, as well as an approach to formal methods tools classification and integrity.

1 Introduction

The SafeFM project [6] aims to provide guidelines on a cost effective approach to using formal methods in the development and assessment of high integrity systems. The project is sponsored under the SafeIT initiative by the UK Department of Trade and Industry and the EPSRC and is a collaboration between AEA Technology, Consultancy Services, the Flight Systems Division of GEC Marconi Avionics and the Formal Methods Group of Royal Holloway, University of London. We believe that SafeFM is at the cutting edge of realistic and practicable use of formal methods in the high integrity system sector.

The focus of the project is on improving current practice rather than the theory of formal methods for high integrity systems. The issues we are concerned with include requirements analysis and specification, the migration of formal methods into existing best practice and the

*Work partially sponsored by EPSRC Grant No GR/H11471 under DTI Project No IED4/1/9013

†Department of Computer Science, Royal Holloway, University of London, Egham Hill, Egham, Surrey TW20 0EX (email *bruno@dcs.rhbnc.ac.uk, victoria@dcs.rhbnc.ac.uk*)

‡AEA Technology, Thomson House, Floor 4, Risley, Warrington, Cheshire, WA3 6AT (email *andrew.boothroyd@aea.orgn.uk, peter.bradley@aea.orgn.uk*)

§GEC-Marconi Avionics Ltd, Instrument Systems Division, Airport Works, Rochester, Kent, ME1 2XX (email *Tim.Boyce@gecm.com, Jonathan.Draper@gecm.com*)

¶ESI Parque Tecnológicó de Zamudio,#204, E-48016, Bilbao, Spain (email *bob.smith@esi.es*) (Work carried out while with GEC-Marconi Avionics Ltd)

assessment of safety critical systems. In particular, we aim to provide guidelines for a cost effective methodology for the specification, development and dependability assessment of complex, safety related systems. Maintaining the practical focus of the project, our work builds upon existing practices in the requirements specification, development and assessment processes as identified in [1, 17]. Our approach is centered on the integration of the development and assessment lifecycles thus supporting existing and emerging safety critical system standards such as the UK MOD DEF STAN 00–56 [14] and INT DEF STAN 00–55 [13] and IEC 65A WG9/WG10 [12]. We, therefore, anticipate that the project results will be useful to organisations aiming to obtain an edge in the development and procurement of dependable computer based systems.

The project is structured in the following themes:

- Adopting a system wide approach to identifying requirements and expressing specifications.
- Supplementing existing best software engineering practice with formal methods.
- Extending existing dependability assessment techniques by building on formal development approaches.

These themes broadly define partner roles although, typically, tasks require substantial partner cooperation. Project work is further made more cohesive by concentrating most tasks on a specific case study, a substantial real world avionics application.

It is the purpose of this paper to report our interim results relating to the first 18 months of the project. In this time progress has been made on all of the above themes. The first section introduces the project case study. We follow this with a discussion of specification coherence including our definition and reasoning strategy. Section 4 continues this theme by recounting our experiences with the application of these principles to the avionics case study. The following section describes our approach to incorporating coherent formal specifications in existing best practice. The next two sections develop the risk assessment theme. In particular we describe work on establishing well defined links between risk analysis and formal specifications, the input of the proof activity to post-development assessment and a method for tool integrity classification together with its application to the SafeFM formal methods process.

2 The SafeFM case study

The project case study [3] has two purposes: First, it ensures that project technology is practical for use in real avionics systems. Second, it acts as a unifier and integrator of the partner activities thus ensuring cohesion amongst the tasks.

The case study is based on a subset of an air data computer (ADC) embedded in a fighter aircraft. The plane is equipped with variable geometry wings which can be swept forward or aft in order to optimise flight characteristics. The ADC also manages further control surfaces which interact with the wing sweep control system.

The case study concentrates on the interactions of the wing sweep with the other control surfaces to optimise the handling of the aircraft. The range of positions required for the optimisation means that some combinations of wingsweep and other surface positions are undesirable. Software limits (as specified by the case study) prevent these combinations. Specific control characteristics depend on various physical parameters such as the altitude and speed of the aircraft and are influenced by commands and modes selected by the pilot. The pilot is not expected to take notice of the control surfaces. The limits and functionality of the interlocks are designed to protect the system from undesirable pilot commands (for example asking the wings to be swept behind the aft limit).

Fault tolerance requirements are also present. In the case study these are limited to the wing sweep control. In order to tolerate mechanical failures, the wing sweep can be controlled by two independent servos. The ADC must detect and signal a possible failure of one of the servos or of the associated actuator and is required to tolerate such a failure. Accordingly, the ADC consists of two different channels each connected to a different wing sweep servo. The architecture is asymmetric, that is a primary channel performs all the ADC functions in normal conditions and a backup with limited functionality takes over when the primary fails. Fault tolerance is not an essential safety requirement but it is included to enhance reliability. Therefore, fault tolerance features must not in any way compromise the safety requirements. The ADC detects and signals possible failures in order to let the pilot try and select the working channel using the reset. It also gives the pilot some indication of problems with the aircraft.

3 Coherent specifications for safety critical systems

Safety critical computing systems are often embedded within larger systems involving a multitude of engineering disciplines. System requirements descriptions can be complex, diverse and large. Building software from such descriptions by extracting the relevant requirements is a non-trivial task which can typically only be undertaken by individuals with domain-specific skills. Ensuring that the software related requirements are consistent and therefore possible to implement is also non-trivial. We have proposed a method of demonstrating the coherence of specifications [11] and we have evaluated it using the project case study [9].

The method is tailored to real-time digital controllers. It is based on the assumption that specifications of such applications can be structured into three broad classes:

- a model of the controlled system,
- a set of functional requirements, and
- a list of non-functional requirements.

The model lists assumptions about the system under control, the functional requirements describe the controller tasks and the non-functional requirements are critical properties typically related to safety, fault tolerance or timeliness.

Given the above specification structure, we define coherence as the combination of two properties:

- *Functional specification consistency*, that is, there are no contradictions in the functional requirements (the system is implementable).
- *Non-functional property satisfaction*, that is, any implementation satisfying the functional requirements also guarantees the non-functional properties (which implies that the non-functional specifications are also consistent).

Demonstrating specification coherence typically involves:

- *Type checking* which establishes for example that there are no out of range values or division by zero errors.
- *Semantic verification* which establishes application-independent properties such as termination and freedom from deadlock [15].
- *Coherence verification* which establishes application-dependent properties such as safety and fault tolerance.

4 Reasoning about safety critical systems specifications

The objective of reasoning about requirements specifications is to establish (and possibly improve) their quality. Reasoning about the ADC specifications established the type consistency of its functional requirements in a straightforward manner. However, and much to our disappointment, the initial results obtained from the case study were negative [9]. Our first attempt to show coherence between the functional and non-functional requirements of the system was, for the most part, unsuccessful. We did succeed in showing that, given perfect wing sweep actuators and servos, the primary channel will never fail; a fairly shallow property.

This inability to discharge the coherence proof obligations was extremely worrying to us. Until that is, we realised that our reasoning efforts were not failing, they were merely entering the realm of reality. The relevance of formal proof in the real world is not dependent upon showing that some specification property holds; aiding the process of discovery and analysis of a complex system, as was indeed the case in our study, comes much closer to fulfilling the role of proof.

We traced the origin of the problem to the form of the case study requirements where the emphasis is on functionality. There is very little information about the controlled system and the safety requirements are not explicit. This is the case with such specifications as the software engineering team typically deals with the software aspects only. But since safety is a property of the system and not the software, formulating and discharging safety proof obligations requires knowledge about the controlled system expressed as assumptions. Such information is not often available to the software engineers since it is not needed in order to develop the software. It is therefore unreasonable to expect the software engineers to discharge such proof obligations given the typical content of the specifications they work from.

In our case, the solution was to consult with the systems engineers who have a much wider view of the system including hardware and software components as well as safety mechanisms. This interaction has helped us clarify the safety requirements and formulate the correct assumptions under which the verification has been reattempted. The resulting proofs of three major safety critical properties (for instance that the backup channel converges to a safe state after it assumes control) as well as a critical evaluation of the work are presented in [10]. The verification which was carried out with the PVS system [7] involved approximately 4,500 lines of PVS, the proofs of 385 lemmas and theorems and required 18 man months of effort.

5 Specifications and structured software development

In order to use formal methods, without completely changing the software development lifecycle, it is necessary to devise an integrated lifecycle combining structured and formal techniques. The lifecycle examined by SafeFM uses VDM in conjunction with Yourdon specifications [2], uses structured design techniques, codes in SPADE Ada Run-Time Kernel (SPARK) Ada, analyses the code with SPARK and tests the code against a test specification produced from an ML animation of the VDM [5, 4].

This lifecycle has been used in our avionics case study. The function considered was originally coded in about 2400 lines of Ada, but only parts of the function were formally specified - 600 lines of Ada were developed from the specification. The formal and informal specifications interacted at two levels: the formal specification described a subset of the complete functionality; and some of the basic functions used by the formal specification were described with a high level of abstraction. The detailed functionality of the abstract functions remained informally specified.

The VDM specification was animated by translating it into ML. This was relatively straightforward because the specification was written in an explicit constructive style. Equivalence

classes and boundary values were identified for each of the inputs. These were combined to produce a large number of test cases (about 15,000). The animation was applied to the list of test cases to produce a list of inputs and outputs. Functions were written to express some of the requirements and these functions were mapped over the list of inputs and outputs to show that the animation (and, by implication, the specification) met the requirements. The animation test cases are also used when the code is tested.

Code was developed informally from the VDM specification, using object based top level design and structure chart notation for the detailed design. The code was statically analysed by the SPARK Examiner for well-formedness and to verify information flow properties. The code was tested using test cases derived from the animation results. A test harness generation tool was used to automate the running and analysis of test cases. Logiscope, a static code analysis and test coverage measurement tool, was used to record metrics such as the percentage of statements executed, in order to help determine test completeness. Despite the use of the formal techniques during development some coding errors were made and these were detected by either the SPARK analysis or the dynamic testing. Further work is being considered regarding the use of the SPARK Theorem Prover to introduce formalism to the process of code verification.

6 Formal methods in risk management

Unusually, SafeFM is interested in formal methods not only as a development approach but also as an aid to safety critical systems risk (where risk is defined as the product of the likelihood of occurrence of an undesirable event by the magnitude of the likely resultant losses) management. Such a perspective is necessary for the practical assessment (and licencing) of systems developed using formal methods. The objectives here are twofold:

- Provide some well-defined link between risk analysis and formal methods.
- Structure proof activity so that it can be easily incorporated in post development assessment activities.

Ambiguities and lack of expressiveness in the semantics of fault trees and related techniques have motivated us to investigate how risk analysis techniques, such as fault trees, can be supported by a formal semantics.

For example, in the case of fault trees, it is not clear whether events should be instantaneous or have a duration, or what timing constraints should be placed on events which are disjuncted or conjuncted together in a fault tree. Expression of base events in a formal notation is generally desirable, to reduce ambiguity. It would also be beneficial to be able to express a time delay which may occur between cause and effect, and whether satisfaction of inputs to a gate condition is a necessary, sufficient, or necessary and sufficient condition that the event above the gate occurs.

We have been able to provide a well defined link here by giving general fault trees a duration calculus semantics which deals successfully with all of the above issues [16].

We are now concentrating on how requirements specifications can be expressed and analysed using an integrated approach of structured analysis, Z, and timed coloured petri nets. We hope that this will provide us with a powerful suite of tools for expressing and analysing fault trees, including their timing requirements. Further plans in this area include the extension of the ideas into other risk analysis techniques such as event trees.

Additional formal proofs were conducted with the objective of identifying problems and solutions in the execution and presentation of tool-supported formal proofs from the assessment point of view. The specific proofs were selected so as to cover a range of types, for example

proof that the system state is correctly described, proof of correctness of system operations, and proof of specific safety properties.

The issues considered were the construction of proofs, the impact of the integrity of the tools on the outcome of the proofs, and the presentation of the proofs such that they are meaningful to an assessor who is possibly unfamiliar with the specific notations involved.

In order to minimise the complexity involved in conducting proofs, a strategy for developing proofs and guidance rules for developing specifications were documented. Other SafeFM work on the project has looked at the feasibility of structuring the specification by separating attributes such as architecture, functional requirements, and safety requirements, which has an impact on the presentation of proofs, which was considered in this part of the work.

Further work planned in this area will extend the scope of the proofs to include reification proofs (proofs between specifications at different levels of abstraction).

7 Formal methods tool classification and integrity

Recent standards, such as INT DEF STAN 00-55, mandate tool support for some processes and this is necessary to make other processes practicable for large developments. The standards require that the tools used have sufficient integrity (for example INT DEF STAN 00-55 mandates that the safety integrity requirements for the tools is established by a hazard and risk assessment according to DEF STAN 00-56).

SafeFM has carried out an investigation into tool support for formal software development [8]. It first classifies the processes in the development supported by tools, then assesses a range of tools against this classification, and finally performs a hazard analysis on an example development methodology. At present no single tool or integrated set of tools supports the entire software development from high level safety requirements to executable object code. Thus, any tool support for development will be incomplete or provided by a number of separate tools. In order to assess the problems this would cause and to see which processes are supported and which are not, it was necessary to classify the stages of the development and consider how errors are introduced and subsequently discovered. The classification is based on five stages of development: safety requirements, specification, design, source code and object code. It considers the processes used to develop each stage from the preceding stages, the processes used to check each stage in isolation, and the processes used to verify and validate each step.

In order to get a feel for how close we are to having an integrated set of tools, a number of tools were assessed against the classification scheme. Three tools were evaluated in order to assess their functionality and five tools (including a generic compiler) were assessed on prior experience. The tools assessment concluded that, although there are tools to support the design, validation and verification of every stage of the development, no single tool (or set of tools) supports all (or even most) of the development. Of the tools assessed, only compilers provided support for the automatic transformation of one level of specification to the next. The tools assessed seemed to fall into three groups: tools that support the analysis of requirements; tools that support the refinement of specifications to source code; and tools that transform source code to object code.

An example development methodology, based on the assessed tools, was analysed to find the integrity requirements of the tools. A fault tree was constructed to model how errors in the final software are introduced and not detected. The processes involved, both manual and tool supported, were assigned weights (ranging from Very High for reviews to Very Low for a tool to have a specific fault that matches a fault in another tool) to represent the likelihood of them introducing or missing errors. As expected, the analysis identified the compiler as a critical tool that should be of the highest integrity. The proof tool used to support safety proofs was surprisingly found not to be critical.

8 Summary

The SafeFM project is based on the premise that formal methods approaches can only fulfill their potential in a practical setting by working together with existing best practice. In the first half of the project we have made much progress towards understanding precisely what is involved in migrating specification and reasoning to real systems. Building on this work, we are now attempting to distill our experiences into a set of specification and verification guidelines for practical use. We also hope to devise a specification calculus tailored to avionics applications. And of course we are continuing the efforts to identify further opportunities of improving best current practice in the development and assessment of safety critical systems by the precision of formal methods.

References

[1] A. Boothroyd, P. Bradley, B. Dutertre, S. Liu, L. Shackleton, B. Smith, and V. Stavridou. Current formal methods applications practices for safety critical systems. Technical Report SafeFM-019-RH-1, SafeFM project, August 1993.

[2] T. Boyce. Formal Techniques in Analysis and Design. Technical Report SafeFM-027-GEC-1, SafeFM project, July 1994.

[3] T. Boyce. SafeFM Case Study Report. Technical Report SafeFM-018-GEC-1, SafeFM project, January 1994.

[4] T. Boyce. Design, Code, Test and Static Analysis. Technical Report SafeFM-040-GEC-1, SafeFM project, April 1995.

[5] T. Boyce. ML Animation and Test Case Generation. Technical Report SafeFM-036-GEC-1, SafeFM project, January 1995.

[6] P. Bradley, L. Shackleton, and V. Stavridou. The SafeFM Project. In F. Redmill, editor, *Proc. of Safety Critical Systems Symposium 93*, pages 168–176. Springer-Verlag, February 1993.

[7] J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. A Tutorial Introduction to PVS. In *WIFT'95 Workshop on Industrial-Strength Formal Specification Techniques*, April 1995.

[8] J. Draper. Tool Support for Formal Development. Technical Report SafeFM-024-GEC-1, SafeFM project, March 1994.

[9] B. Dutertre. Case Study Coherent Specifications. Technical Report SafeFM-050-RH-1, SafeFM project, July 1994.

[10] B. Dutertre. Coherent Requirements of the SafeFM Case Study. Technical Report SafeFM-050-RH-2, SafeFM project, September 1995.

[11] B. Dutertre and V. Stavridou. Coherent Requirements Specifications for Safety-critical Real Time Systems. Technical Report SafeFM-020-RH-1, SafeFM project, February 1994.

[12] Draft IEC Standard 1508 - Functional Safety: Safety-related Systems, April 1995. International Electrotechnical Commission, Technical Committee no. 65, Working Group 9/10 (WG9/10), IEC 65A.

[13] *The Procurement of Safety Critical Software in Defence Equipment.* Ministry of Defence, Directorate of Standardization, Kentigern House, Glasgow, UK. First Draft Defence Standard 00-55, Issue 2.

[14] *Hazard Analysis and Safety Classification of Computer and Programmable Electronic System of Defence Equipment.* Ministry of Defence, Directorate of Standardization, Kentigern House, Glasgow, UK. Second Draft Defence Standard 00-56, Issue 2.

[15] D. L. Parnas. Some Theorems we Should Prove. In *Proc. of 1993 International Meeting on Higher Order Logic Theorem Proving and Its Applications*, pages 156–163. The University of British Columbia, Vancouver, BC, August 1993.

[16] K.M. Hansen, A.P. Ravn and V. Stavridou. Linking Fault Trees to Software Specifications. In *Colloquium on Analysis of Requirements for Software Intensive Systems.* DRA Malvern, 19-20 May 1993.

[17] V. Stavridou, A. Boothroyd, P. Bradley, B. Dutertre, L. Shackleton, and R. Smith. Formal Methods and Safety Critical Systems in Practice. Submitted for publication to *High Integrity Systems* 1995.