A Practical Approach for Closed Systems Formal Verification Using Event-B

Brett Bicknell[1], Jose Reis[1], Michael Butler[2], John Colley[2], and Colin Snook[2]

[1] Critical Software Technologies Ltd, 2 Venture Road, Southampton Science Park, Southampton, SO16 7NP, United Kingdom
jreis@critical-software.co.uk,
WWW home page: http://www.critical-software.co.uk
[2] Universitity of Southampton, Electronics and Computer Science, SO17 1BJ, United Kingdom

Abstract. Assurance of high integrity systems based on closed systems is a challenge that becomes difficult to overcome when a classical testing approach is used; in particular the evidence generated from a classical testing approach may not meet the objectives of rigorous standards. This paper presents a new approach for the formal verification of closed systems, in particular commercial off the shelf (COTS) products. The approach brings together the formal language Event-B, mathematical proof theory and the Rodin toolset and provides the mechanism for creating abstract models of closed systems and to then verify these system properties against operational requirements. From an industrial perspective this approach represents a step change in the use and successful integration of closed systems; using formal methods to guarantee their integration and functionality. The outcome of the proof of concept will provide a solution that will increase the level of confidence on complex system of system solutions containing closed systems. Moreover, it will support the production of safety-cases by providing formal proofs of a system's correctness.

Keywords: closed systems, COTS, Event-B, Rodin, formal verification, virtualisation, VMWare

1 Context

Closed Systems (CS) are becoming more common in the safety critical domain and in particular within the military domain. As the name connotes a closed system is a solution which is available as a black box where access to internal modules and design artifacts is not possible. In the context of this paper we will refer to a closed system as a software based system and of which can refer to either of the following contexts:

1. Systems developed by a third party that are commercially available to the public domain as readymade solutions; also known as COTS

2. Systems developed by a third party that are not commercially available but due to legal, export controls or other commercial reasons are not open to the system integrator

The justifications provided by system integrators for adopting a CS product as opposed to a bespoke solution are typically:

- Adoption of state of the art technology : The supplier of a CS has the budget to invest in R&D and innovative methods to provide state of the art technology. Furthermore these suppliers tend to invest in and continuously develop the product to stay competitive for longer
- Cost reduction : It is debatable whether a CS product offers a significant cost reduction or not, because the CS product may not be integrated seamlessly with the complete end system. If instead there are no significant issues with the integration of the CS, the end cost will be less than if the same solution was developed from scratch. Another key factor is that having the product on an open market pushes suppliers to offer competitive prices
- Reliability : In principle CS products have been tested thoroughly by the supplier, in that the solution is deemed reliable in particular contexts. The caveat is that the customer may have conditions that the solution has not been tested under and this becomes an issue, in particular when the CS is used in a safety critical system

The rollout of a CS product in a safety critical domain is not a straightforward activity as the assumptions made about the CS product may not hold when the product is integrated and tested with the rest of the system. It often happens that the CS product does not operate as expected in the specific environment or that some of the system requirements are not satisfied by the closed system. Furthermore the Verification and Validation (V&V) of Closed Systems is particularly difficult to perform because in most cases there is no design or requirements documentation available in the public domain, nor has the evidence of V&V activities performed by the supplier been made public. Determining the conditions and environment under which the CS product was validated is not simple. The non-existence of this data makes it more difficult to build a case for the reliability and safety of the end product.

Where all this might not be vital in a non-safety critical domain, it is particularly relevant in a safety critical domain where, for instance, the DO-178 standard requires evidence of V&V activities to meet specific objectives.

2 Existing Approaches

2.1 Integration Testing and Prototyping

The validation of CS based solutions typically starts as soon as the CS product is made available to the system integrator. First the product is taken through a prototyping stage with parts of the system being integrated with the CS products.

This stage validates some functional integration aspects; it does not necessarily address safety or reliability properties of the system as the focus is instead on validating the functional integration with the end system. The prototyping stage may lead to the identification of issues with the CS that require customizations, of which in some cases the supplier of the CS product can be open to fixing. This stage can also identify limitations which could lead to either rejecting the CS product or the system integrator having to identify workarounds.

Once the various elements of the solution are amenable to integration the system is formally built and only then can it go through integration testing. The approach used for testing is driven by the set of requirements including safety requirements, use cases, the test environment and last but not least the time available to perform the testing.

There are several risks associated with this approach:

– The identification of issues and the need for workarounds might come late in the lifecycle when there is no other option but to develop bespoke solutions. This impacts on the cost and schedule of the end system
– A limited test coverage as a result of the limited time available for testing and a limited test environment which does not allow for simulating the real in-service environment. This impacts on the confidence in the end solution and could result in a system deployed with untested states and an impact on the ability to generate a sound safety case
– As safety can be seen as an emergent property of the integrated system [menon2009interim], the safety analysis, which could result in the need for significant re-design, occurs too late in the verification process

2.2 Fault Injection Testing

Fault injection testing is another approach that aims to address scenarios where there is insufficient evidence to demonstrate that the CS will not fail unacceptably or where the CS does not satisfy safety requirements. The primary objective of fault injection testing is to understand whether additional fault detection, isolation and recovery (FDIR) mechanisms are required or if the existing FDIR mechanisms are satisfactory. An example of a successful application of the method is described in [MSM$^+$02].

The key risks associated with this method are:

– The APIs may not be sufficiently well documented resulting in additional effort to effectively customise the test environment and integrate it with the target system
– The results produced are constrained by the number of fault scenarios generated and as a consequence the evidence generated may be inadequate to demonstrate the robustness of the target system
– The fault injection tool may not provide all of the interfaces required to integrate with the equipment under test and as a consequence it may not be possible to inject faults in the target system

– Potential failures caused by unexpected component interactions in the integrated system are not addressed by this approach

There are a number of other approaches that can be used to facilitate the verification and validation of CS based solutions, such as using pre-existing evidence or reverse engineering. For further details see [MHM09].

3 Solution Description

This paper outlines an approach that we have been applying to a real industrial project using the Event-B formal method. The system being analysed uses a number of complex static configurations and we were able to specify general requirements on these configurations in Event-B, and verify specific configurations using model checking and mathematical proofs. We also use refinement to map a high level model of the requirements to a lower level model that represents an abstraction of the CS product. Our experience to date suggests this approach encourages a deeper analysis of the system requirements and how these relate to the characteristics of the CS product than a less formal approach. The formal models and verification results also contribute to the safety case for the system.

The work on the case study is still in an intermediate stage, yet enough work has been completed to draw preliminary conclusions on the method and issues faced.

3.1 Tools Description

The case study uses the Event-B language, which is a formal language built upon set theory and mathematical proof, and is an evolution of the industrial-strength B method. This is in combination with the Rodin toolset, which is an Eclipse-based open source development environment for Event-B models [ABH$^+$10]. Rodin includes proof obligation generation and automated proof tools along with a range of model-checking plug-ins. To date, Rodin has mostly been used in a research environment, yet using real industrial case studies.

The extensions to Rodin which are utilised in this case study are UML-B [SB08] - a UML-like graphical front end for Event-B which provides strong support for model refinement - and ProB [LB08], a plug-in which supports simulation and model-checking of both Event-B and UML-B models.

3.2 Case Study Overview

The system to be verified through the case study is a shared computer environment (SCE) which consists of hardware, virtualization middleware and applications. The hardware is composed of multiple servers, and the virtualization is provided by VMWare - a closed system - which enables multiple applications to be running on the same server and provides dynamic load-balancing and resource reallocation across the system.

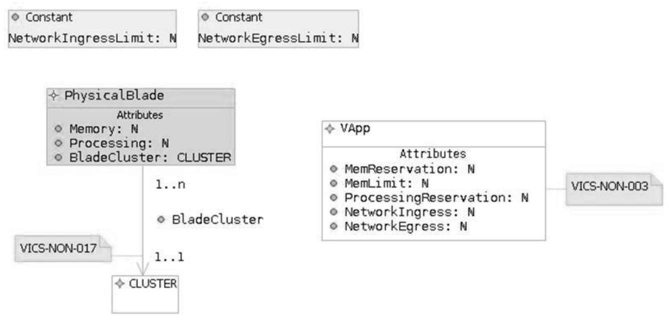

Fig. 1. An example of a UML-B model used in the case study

The abstract design for the system is broken down into logical servers, which represent groupings of applications that should be running on the same server. The hierarchy of the applications is introduced by the classification of each application to a function or sub-function; this allows for functions to span multiple logical servers if they have sub-functions running on each. In relation to VMWare, each function and sub-function corresponds to a Virtual Appliance (VApp) and can have multiple Virtual Machines (VMs). Each SCE operates according to a fixed number of static configurations mapping functions and sub-functions to logical servers. In this way it is possible to define configurations for multiple SCEs, a single SCE (should one or more of the SCEs fail), or minimum safe configurations (so that the critical applications can continue running in the event of multiple server failures in the same SCE)

The properties of the system that can be verified formally are broken down into two phases; the static verification of each configuration, and the dynamic verification of the design requirements with the behavior of VMWare. The static phase of the verification involves proving that the configuration adheres to the requirements within the design - for instance, that a function assigned to a single logical server should not have any sub-functions - but also that the memory, processing and network resources assigned to the sum of the functions and sub-functions running on each logical server do not exceed the limits on a single server. VMWare can control the memory and processing resources; however, if the overall limits for the server are exceeded then it cannot guarantee that all of the applications will run correctly. Perhaps more critically, VMWare has no control over the network usage; so it is imperative that the network resource use is checked before each configuration is run.

In each of the static cases the formal verification is achieved by modelling the requirements as axioms in the model and then inserting the mappings specific to the configuration into the model. To verify the static design requirements we used ProB to check that the axioms are consistent, i.e. that ProB can find sets and constants that satisfy the axioms. By using ProBs constraint solving abilities, we discovered several modelling errors when ProB could not find a

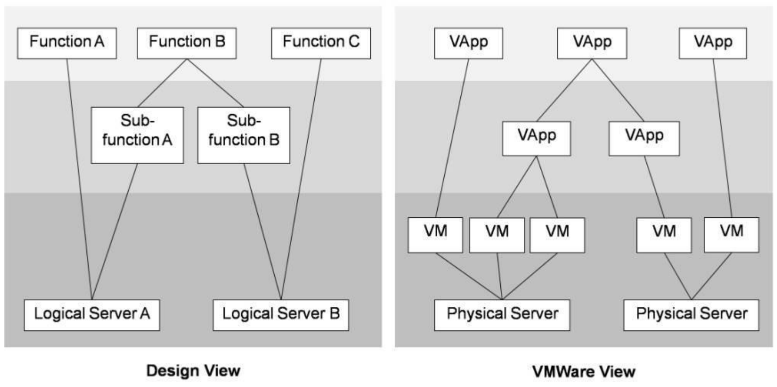

Fig. 2. Comparison of VMWare and design views

solution that satisfied the axioms. We then verified that an example configuration (also defined using axioms) was consistent in ProB and therefore satisfied the axiomatic constraints on configurations such as affinity rules and function allocations.

The advantage of using a model checker in this case is clear; if new configurations are designed, or existing configurations are changed slightly by adding new functions, then a static check can be simply and quickly performed on the configuration in an automated fashion without having to manually recalculate any of the resource limits or check that the mappings are valid.

To verify that an example configuration obeys the resource requirements we used the Rodin provers. The resource rules were expressed as a theorem which, if the configuration is valid, should follow from the axioms describing the resourcing required by the configuration. This involved time consuming interactive proof, even for a simple configuration. As a result we have proposed a new proof rule which will be added to the Rodin provers, thereby decreasing the time needed to prove future configurations.

The dynamic verification involves demonstrating that the combination of the system and VMWare behaves in such a way that the system design requirements are not invalidated, as these requirements are not inherent within VMWare. More specifically, there are certain affinity rules within the design, such as groupings of applications which have to be running on the same server, which could easily be invalidated by VMWare if it decides to reallocate applications due to an increased load on one or more of the servers. The potential issues are even more apparent when server failures are considered; in this case entire logical servers have to be reallocated or the configuration has to be changed, all by VMWare whilst retaining the design requirements and operational capacity of the system. The formal approach here, although not yet complete, will be to model the dynamic behavior of VMWare in Event-B, and verify this using the Rodin provers against

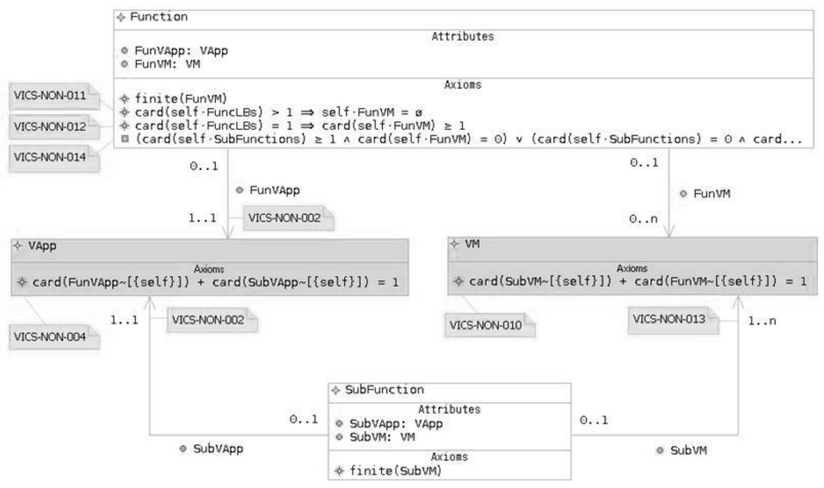

Fig. 3. Further example of a UML-B refinement step in the case study

the existing axioms in the static verification representing the design and resource requirements.

The key limitations to the approach are:

– The complete CS has to be modelled and thus assumptions have to be made about its behaviour. This is in fact what determines the scope of the modelling; for instance, it is assumed for the dynamic behaviour that the memory reallocation and low-level processes work reliably. However it is important to keep in mind that these are the type of issues that, as mentioned earlier, are often verified during the development of the CS.
– Another known limitation to this approach is inherent to the Event-B language and is due to the restricted set of requirements that are amenable to the language. Only a subset of the requirements could be modelled; it is not possible to model requirements concerning the performance of the system, for instance. This is not to say that this is not possible with other formal languages, however.
– A further limitation which has become apparent through the work is that the requirements presented are often in an unsuitable form for formal modeling, and consist of a mix of high and low levels. During the case study another, intermediate, requirements document had to be produced to fill the gap between the initial requirements and the formal model.

The modelling process is reliant on the participation of the customer; at each step in the process models were produced, which raised questions, and after more information was gained the models were developed further. A benefit of using a formal method such as Event-B is that it encourages deep requirements analysis due to the completeness of requirements necessary for formal modeling; before and during the modelling missing and inconsistent requirements were found, most of which were corrected or included in the intermediate requirements document. A further benefit of the Event-B approach described is that it encourages the investigation, discovery and modelling of safety constraints much earlier in the verification process than traditional methods.

4 Preliminary Conclusions

The work on the case study is still in an intermediate stage, yet the following preliminary conclusions can be drawn:

– Static properties of VMs running in VMWare can be formalised using the Event-B method and verified using the Rodin toolset
– Event-B is a suitable method to verify properties which are not managed by VMWare but which are critical and have to be validated before running the virtualised applications
– The Event-B method offers the mechanisms to abstract specific details pertinent to the CS and focus on a sub-set of properties which are complex to verify using a classical testing approach
– Event-B models of VMWare can be reused and expanded with more detail if required
– We found UML-B's class diagram notation very suitable for modelling this system. UML-B allowed us to construct a model very quickly and to explore different modelling choices, along with providing a mechanism to visualise the model and communicate it to others

Further work is going to be performed to formally verify the design requirements with the behavior of VMWare.

References

[ABH+10] J.R. Abrial, M. Butler, S. Hallerstede, T.S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in event-b. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(6):447–466, 2010.

[LB08] M. Leuschel and M. Butler. Prob: an automated analysis toolset for the b method. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(2):185–203, 2008.

[MHM09] C. Menon, R. Hawkins, and J. McDermid. Interim standard of best practice on software in the context of ds 00-56 issue 4. *SSEI, University of York, Standard of Best Practice*, (1), 2009.

[MSM+02] H. Madeira, R.R. Some, F. Moreira, D. Costa, and D. Rennels. Experimental evaluation of a cots system for space applications. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 325–330. IEEE, 2002.

[SB08] Colin Snook and Michael Butler. Uml-b and event-b: an integration of languages and tools. In *The IASTED International Conference on Software Engineering - SE2008*, February 2008. Event Dates: 12-14th February 2008.