An overview of the SystemB collaborative project

Edward Turner, Helen Treharne¹, Neil Evans²

¹Department of Computing, University of Surrey

²AWE plc, Aldermaston, Reading.

Edward.Turner, H.Treharne, N.Evans @surrey.ac.uk

Abstract

Much research work has been done on linking UML and formal methods but few have focused on using formal methods to check the integrity of UML models so that the models can be verified. A current collaborative project between the University of Surrey and AWE plc is investigating the underpinning of xUML platform independent models with CSP || B specifications. Our position paper presents an overview of the project and demonstrates an opportunity for applying formal methods in industry. One of the project's aims is to support reasoning about concurrent xUML state machines. xUML is a subset of UML which provides support for code generation. AWE has already developed mappings between xUML and SPARK Ada and our current work will provide further techniques in order to ensure the safety of their software systems. UML has become the language of choice for systems engineers at AWE and support for formal techniques is necessary.

Keywords: B, CSP, MDA, Automatic Verification, UML

1. INTRODUCTION

We have recently started work on a two year collaborative project, September 2007 - August 2009, involving the Department of Computing, University of Surrey and AWE plc, and this position paper provides an overview of the project. Our motivation is to develop an approach in which the formal representation of a UML [11] model can be used to verify the integrity of the model and assess whether the model is capturing desirable behaviour. The models of interest to us are safety critical ones and therefore behavioural properties including deadlock, livelock and other safety properties are required to be verified. Current industrial practice includes running numerous simulations on UML models but no verification is carried out in order to ensure an acceptable level of safety for a particular system.

UML is widely used within AWE, and a subset of UML referred to as Executable UML (xUML) [10] is one of the preferred methods. xUML offers the capability of being able to support code generation into C, C++ or Java from platform independent models. Systems engineers have also been working alongside Kennedy Carter, the suppliers of the xUML toolsuite, to develop Spark Ada translators from xUML.

An xUML model of a system may comprise of six parts:

1. Use Case Diagrams capture the requirements of the system,
2. Domain Models partition the system into separate parts,
3. Sequence Diagrams define the interfaces of the various domains,
4. Class Diagrams specify the classes in each domain,
5. Collaboration Diagrams define the class interfaces,
6. State Machines specify the behaviour of each class.

An xUML model is validated by review and by running simulations on it. The simulations are conducted on platform independent models prior to code generation. A simulation requires

FIGURE 1: Overview of integrated system development methodology

populating a model with typical objects and appropriate values, and running different test methods which have to be defined. For example, in order to simulate an xUML model of a barrier system with two lights, objects related to the lights and the barrier would have to be created. Then the sequence of the barrier's operation in relation to the lights displaying red or green would be defined as test methods so that the model's state machines are exercised.

Our project is split up into a number of phases. In the first instance we will focus on state machines and class diagrams as defined in xUML, and develop an approach to construct CSP || B [14] specifications from particular xUML models. CSP || B is an approach that has been defined to integrate the state-based method B [1] and the process algebra CSP [6]. The approach has focused on identifying compositional verification techniques so that a specification can be verified in smaller parts. We have developed several examples, including a file transfer protocol [4], which demonstrate the applicability of our compositional techniques. Preliminary investigations show that a new style of CSP || B specification is required to be defined so that CSP || B can be tailored to form a suitable formal representation of xUML state machines [15]. The investigations also show that we can still apply our compositional verification techniques to reason about the concurrent behaviour of the state machines. In the barrier system we would want to verify that the barrier is always down when the light displays red. The next section of the paper provides more details of the initial phase of the project and the current challenges we have identified, and Section 3 details other possible research directions within the other phases of the project.

Research which combines formal and informal notations is not new. For example, there have been attempts to provide a formal semantics for UML. The underpinning has either been targeted at the UML meta level or at the level of the various UML diagrams themselves, mainly class diagrams, sequence diagrams, and state diagrams. For example, the B-Method has been used to provide a formal description of the class meta model [8]. The work of Fitelson et. al [5] focuses on using predicates to describe the associations in class diagrams and Börger et al. [2, 3] have concentrated on formalising UML state machines using ASM. Research has also been conducted in which UML has been used to provide a graphical front end for formal notations. For example UML-B [12] uses a UML-like notation and the B method as an action and constraint language with the goal of deriving a B formal specification.

By the end of the project we hope to have made good progress towards our longer term aim of merging xUML and formal methods into an integrated systems development methodology so that CSP || B specifications can be generated automatically from suitable xUML platform independent models. We will be constrained by how xUML models are used in practice. This constraint is challenging, but exciting, because the end result will be an approach which addresses the particular concerns of our industrial collaborator. Thus, we envisage a development methodology as illustrated in Figure 1.

FIGURE 2: Two state machines

2. INITIAL PHASE

In xUML state machines specify the behaviour of each class. Each class can have at most one state machine. Classes are defined in class diagrams and there is a class diagram per domain of interest. In an xUML model there may be several domains. Domain modelling provides a way of partitioning a system into several parts so that different functionality can be considered separately. For example, in the barrier system outlined above, we could envisage a domain to deal with payment and another to deal with the actual function of the barrier. In the initial phase of the project we will focus on what it means to translate the information captured in a single domain to a formal model.

Consider the example in Figure 2 which shows two state machines. A larger example is presented in [15]. The relationship *R1* represents the association between a *Barrier* and *Light* class and would be captured in a class diagram where a barrier is related to two lights.

Each state can have an entry action which enables the definition of a sequence of ASL commands. ASL is the language within xUML which allows us to manipulate the UML object model. Actions are executed upon receipt of a signal or as a result of object creation or deletion. A fragment of ASL is shown in the entry action of *Barrier.up*, in Figure 2. The local variable *myLights* represents the instance handles of the light objects. The first assignment of the entry action assigns a value to *myLights*. These instance handles can be found by navigating from *this* through *R1* (*this* -> *R1* in the figure), where *this* refers to the instance handle of the object associated with the *Barrier* class.

Signals enable the transition between states. It is possible to send a signal to a specified object using the “generate <<signalname>> to <<instancehandle>>” construct. For example, if the state of a light instance *l* is *Light.red* then the *switchToGreenSignal* signal is sent to that instance. If the state machine of a light object is in the *Light.red* state and has done its entry action then the processing of the *switchToGreenSignal* allows us to move from the *red* state to the *green* state.

In the project we will investigate how different states, e.g. *Barrier.down* and *Barrier.up*, map to pairs of CSP processes, one which captures the behaviour of the entry action and one which receives signals in order to perform transitions to other states. In order to model how the signals are processed we will need to provide an appropriate abstraction of the execution models available within xUML. There are three models with different levels of concurrency. Any analysis will need to be conducted in the context of an appropriate execution model.

During the execution of an entry action objects are queried and their attributes may be updated. We will be using B machines to capture the definitions of objects and their attributes. Therefore, several of the CSP events which define the behaviour of an entry action will correspond to B operations. Our aim will be to come up with general specification patterns which can be represented in CSP and in B in order to translate the ASL commands within entry actions. Section 2.1 notes some of the challenges we have currently identified.

Once we have generated a CSP || B specification our analysis will focus on two kinds of properties. Firstly, we will check whether a signal is possible when the xUML model is in specific concurrent states. It enables us to highlight any undesirable behaviour within the model. Secondly, we will analyse the integrity of attribute values when the xUML is in specific concurrent states. It ensures that the system is modifying and accessing its state safely. The formal analysis is based on examining combinations of concurrent states from the state machines. Identifying these combinations by hand will not be acceptable in practice and therefore we need to explore how to automate their creation. We also need to investigate how the required properties of an xUML model can be automatically generated as CSP processes in order to be used in FDR refinement checks or as LTL properties using ProB. Analysing system behaviour from concurrent states is not possible within the xUML tools available.

From the above it is clear we need to automate two aspects. We need to automatically produce CSP || B specifications and also the analysis checks. We will develop a translator in the xUML toolsuite and use the same framework for code generation as has been used for other programming languages. The framework involves adding code generating statements at the xUML meta model level. Then, in order to translate a particular platform independent model, the meta model is traversed to extract the corresponding CSP || B specification for that particular model. Again we will be attempting to do this in two stages. Initially, we will produce only CSP specifications. This means that we can focus on the executable domain within the xUML meta model and a small part of the ASL domain meta model. Then we will move on to producing CSP || B specifications which will involve greater use of the ASL domain meta model.

2.1. Challenges of current approach

The ASL language is relational in style but also includes programming-like statements including loops and supports direct access to objects. Modelling such low level constructs in a formal model is not straightforward. For example, consider the state machine of object X and one of its entry actions. The action can iterate over many objects which are related to X, and update the attributes of those objects. This direct access of attributes means that object interference is possible; consequently ensuring the integrity of attribute values is hard. In the Kennedy Carter code generation tools access to objects is controlled by locking mechanisms and we need to consider whether we need abstractions of such mechanisms in our formal specifications.

Currently, we have only investigated the translation of object/class methods as simple B assignments. However, xUML, allows iteration statements and this introduces problems when we translate methods to B. Therefore, we need to assess whether mapping methods to B operations is a valid approach. It works in the simplest of cases but it may not scale. Other approaches have examined the modelling of state machines from a more abstract point of view, including [9], and it may be more appropriate to consider the pre and post conditions of a state rather than individual statements within an entry action.

3. OTHER PROJECT DIRECTIONS

The demonstrator in the initial phase is tied closely to the xUML tools. We will also investigate how to generate CSP || B specifications from UML statecharts using the Epsilon platform developed in York [7]. The platform supports model management and is based in Eclipse. The Epsilon platform offers several target-specific languages including EGL for model transformations. Mapping UML state diagram models to CSP || B specifications is an example of a model transformation. We will expand our scope to include UML state diagrams and rather than xUML state diagrams. This will enable us to use existing UML meta models. Then we will be able to investigate reusing the patterns identified in the initial phase and encode them directly in EGL. This will be done in close collaboration with researchers from York. The benefit of this research is that it will allow us to progress towards defining a CSP || B meta model so that CSP || B specifications can be defined as an instance of its own meta model and that the transformation from an informal UML model to

CSP || B specification is then achieved at the meta level rather than by augmenting a UML meta model with code generation information.

Acknowledgments The authors are grateful to Steve Schneider, Ian Wilkie, Chris Raistrick, Richard Paige and Phil Brooke for useful discussions. Thanks also to the reviewers for bringing other related work to our attention.

REFERENCES

- [1] Abrial, J-R.: *The B Book: Assigning Programs to Meanings*, CUP (1996).
- [2] Börger, E., Cavarra, A., Riccobene, E.: *On formalizing UML state machines using ASM*. Information & Software Technology, 46(5):287-292, 2004.
- [3] Cavarra, A., Riccobene, E., Scandurra, P.: *A framework to Simulate UML Models: Moving from a Semi-formal to a Formal Environment*. Proceedings of the 2004 ACM Symposium on Applied Computing(SAC), pp 1519-1523, 2004.
- [4] Evans, N., Treharne, H.: *Investigating a File Transfer Protocol Using CSP and B*. SoSym Journal, 2005.
- [5] Faitelson, D., Welch, J., Davies, J.: *From predicates to programs: the semantics of a method language*. Proceedings of SBMF 2005. ENTCS, 2005.
- [6] Hoare, C. A. R.: *Communicating Sequential Processes*. Prentice Hall, 1985.
- [7] Kolovos, D.S., Paige, R.F., and Polack, F.A.C.: *Epsilon Development Tools for Eclipse*, Proc. Eclipse Summit 2006, Esslingen, Germany, October 2006.
- [8] Laleau, R., Polack, F.: *Coming and going from UML to B: a proposal to support traceability in rigorous IS development*. ZB2002 Formal Specification and Development in Z and B, LNCS, 2272, pages 517-534, 2002.
- [9] Möller M., Olderog E-R., Rasch H., Wehrheim H.: *Integrating a formal method into a software engineering process with UML and Java*, Formal Aspects of Computing, 2007.
- [10] Raistrick, C., Francis, P., Wright, J., Carter, C., Wilkie, I.: *Model Driven Architecture with Executable UML*. Cambridge University Press, 2004.
- [11] Rumbaugh, J., Jacobson, I., Booch, G.: *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.
- [12] Snook, C., Butler, M.: *UML-B: Formal modelling and design aided by UML*. ACM Transactions on Software Engineering and Methodology, 2006.
- [13] Schneider, S.: *Concurrent and Real-Time Systems: the CSP Approach*, Wiley, 1999.
- [14] Schneider, S., Treharne, H.: *CSP Theorems for Communicating B machines*, FACS 17(4) 2004.
- [15] Treharne, H., Schneider, S., Grant, N., Evans, N., Ifill, W.: *A step towards merging xUML and CSP || B*, Dagstuhl workshop on Rigorous Methods for Software Construction and Analysis, LNCS Festschrift, (to appear).