ABZ2008 Conference

Short Papers

September 16-18, 2008

BCS London Offices, Covent Garden, London, UK

Egon Börger
Michael Butler
Jonathan Bowen
Paul Boca

ABZ2008 Short Papers

Page

Integrating Z Into Large Projects
Tools and Techniques

Anthony Hall

Overview

This paper addresses the problem of using Z to specify large software intensive systems. It is addressed particularly at safety, security or business-critical systems that require a rich language like Z to specify complex data and behaviour. Examples of such systems include air traffic control [1] and financial systems [2]. I am particularly concerned here with the process of writing a system specification.

I have claimed [3] that this process is beneficial in itself regardless of any subsequent analysis or proof. The benefits are not automatic, however. The specification is only useful if

1. it is written as part of the requirements elicitation and analysis process;
2. it is comprehensible to all the stakeholders;
3. it is grounded in the application domain;
4. it expresses all the concepts that are relevant;
5. it is integrated into the rest of the development and verification process.

On a purely practical level, this implies that we want Z to be part of the ordinary documents that are used every day on the project. That means, in practice, that it has to be integrated into Microsoft Word. So I first describe a tool for writing and checking Z within the Word environment. That is only the start, however: we also need a process for writing the specification and guidelines for its structure. I will describe some progress we have made towards these goals.

A Z Tool for Microsoft Word

The de facto standard for writing Z is the LaTeX mark up first introduced by Spivey and now incorporated into the Z standard. Industry, however, does not use LaTeX: it uses Microsoft Word. I have developed, with help from several colleagues, a package of Z tools for Word. This is in day to day use on a large project and is being made freely available at http://sourceforge.net/projects/zwordtools.

The tool includes:

1. styles for laying out schemas and other Z paragraphs;
2. a Unicode font that includes all the Z symbols and is visually compatible with Times New Roman
3. automatic layout of Z paragraphs like the LaTeX equivalent, with italic text, formatted keywords and so on: for example

```
_BirthdayBook____________________________
| known : ℙ Name
| birthday : Name ⇸ Date
|_______________________
| known = dom birthday
|________________________________________
```

4. the ability to enter symbols from a palette or (for died in the wool LaTeX hackers) typing in the markup;
5. one-click typechecking, with errors highlighted in the Word document;
6. generation of indexes and cross-references to definition and use of Z names;
7. the ability to hide the Z completely so the document can be used by maths-phobic readers;
8. miscellaneous tools such as checking matching brackets.

The intention of the tool is

- to lower the barrier to the uptake of Z by removing at least one obstacle, the need to learn another document production method;
- to allow easy integration of Z with natural language, diagrams, tables and other notations relevant to the domain;
- to encourage incremental development of Z specifications by allowing frequent typechecking;
- to encourage the writing of good natural language by producing documents with the mathematics hidden.

The tool currently uses Mike Spivey's fuzz as its underlying typechecking engine. It works by exporting LaTeX mark up and importing the fuzz error report: the intention is that this mechanism could be used with other tools, in particular tools supporting the Z standard. Typechecking does not require declaration before use and works across multiple documents: for example an operations document can be checked against a separate data model document.

Writing Literate Z

It is crucial to realise that in a Z specification, the mathematics is subsidiary to the natural language. A piece of mathematics makes no sense unless we know the intended meaning of each construct. We therefore enforce a rule that an English description must precede the corresponding Z. The English and the Z are complementary: the English describes the relevant real-world concept and explains the meaning of every term in the maths; the maths makes precise the relationships between the terms defined in the English. Note in particular that there is no rule like "In case of a discrepancy between English and Z, the Z takes precedence": rather, the rule is that such a discrepancy is an error which must be corrected.

Data Model

Every schema that defines part of the system state is preceded by an explanation of what properties of the real world are being described. Here is a small example:

Attribute Name	**Definition**
position	The x and y co-ordinates in system coordinates
smoothedAltitude	The uncorrected altitude from the mode C in feet.

Track

Position
smoothedAltitude : *optional Altitude*

Altitude is a natural number that represents uncorrected height above mean sea level in feet. It is the height that an altimeter calibrated to a pressure of 1013.2 millibars at sea level would read. Its correspondence with physical height depends on the current pressure.

$$\mathit{Altitude} == \mathbb{N}$$

Operations

Here is an example of an operation description in the same style. Again the English stands alone and explains the real world meaning of the operation.

Issuing a clearance updates the known clearances.

Inputs are

controller?:	the controller who is issuing the clearance
flight?:	the flight for which the clearance is being issued
clearance?	the new clearance.

The operation is only allowed if the conditions described in *IssueClearanceAvailable* are met. In that case, the clearance is recorded as issued by the controller.

$$\begin{array}{|l}
\multicolumn{1}{l}{\_\mathit{IssueClearance}\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_} \\
\Delta\, \mathit{Clearances} \\
\mathit{controller}? : \mathit{CONTROLLER} \\
\mathit{flight}? : \mathit{FLIGHT} \\
\mathit{clearance}? : \mathit{CLEARANCE} \\
\hline
\mathit{IssueClearanceAvailable} \Rightarrow \mathit{clearanceIssuedBy}' = \\
\qquad\qquad \mathit{clearanceIssuedBy} \cup \{\mathit{clearance}? \mapsto \mathit{controller}?\} \\
\hline
\end{array}$$

ⓘ The new clearance is guaranteed to be unique so *clearanceIssuedBy* is guaranteed to remain functional under this update.

The last paragraph is an example of a *Z Comment*. This is not to explain the real world meaning, but rather to explain technicalities of the Z. If you hide the mathematics the Z comment is also hidden, since it is of no interest to readers of the English.

Summary

If we want to use Z to write an overall system specification, we need to integrate it into a rich set of documents written in natural language and domain-specific notations. These documents must be easy to write and read by non-mathematicians. I have described a tool and sketched a set of rules aimed at achieving this. There is much more to do, both in improving the tool and codifying the process for different domains.

References

1. Anthony Hall, *Using Formal Methods to Develop an ATC Information System*, IEEE Software, March 1996, pp 66-76.
2. Anthony Hall and Roderick Chapman, *Correctness by Construction: Developing a Commercial Secure System*, IEEE Software, Jan/Feb 2002, pp18 – 25.
3. Anthony Hall, *Seven Myths of Formal Methods*, IEEE Software, September 1990, pp 11-19.

Stability of Real-Time Abstract State Machines under Desynchronization

J. Cohen[1] **A. Slissenko**[1♯]
E-mail: {j.cohen,slissenko}@univ-paris12.fr
Laboratory for Algorithmics, Complexity and Logic,
University Paris-East (Paris 12), France

Abstract. We study the stability of real-time multi-agent Abstract State Machines (ASM) under desynchronization. Real time constraints are defined as linear inequalities with rational coefficients over current time CT and real-valued internal functions. There are bounded non-deterministic delays between actions. Our goal is to give sufficient conditions under which after relaxing the delays we get an ASM (the program remains the same) whose set of runs is approximately bisimular to the set of runs of the initial ASM.

Introduction. In our paper [1] we give sufficient conditions that permit to implement a real-time ASM with instantaneous actions (IA-ASM) by an ASM with delayed actions (DA-ASM) with approximate bisimulation of runs. The time is continuous and time constraints are linear inequalities with rational coefficients. (The problems we study are as well relevant for discrete time, but this case is simpler.) The inputs are predicates and the current time CT. As IA-ASM we consider ASM whose programs are blocks of **if** *guard* **then** *blockOfUpdates*. We take a straightforward implementations by DA-ASM with bounded delays. Namely, such an implementation work by 2 phases: backup phase memorizes the values of functions, and update phase makes the updates of the initial IA-ASM using the backed up values. Such an implementation implies shifts of time instants and, consequently, of the values of the real-valued functions. The approximation of runs is determined by 2 positive parameters (ε, η), where ε bounds time shifts, and η bounds the deviations of real-valued functions. We introduce a notion of (ε, η)-sturdy IA-ASM, and prove that the implementation of any such IA-ASM gives an DA-ASM with (ε, η)-approximately bisimular runs if the delay satisfies constraints described in terms of (ε, η), number of real-valued functions, the maximum of absolute values of coefficients, the number of summands in linear inequalities and the *update number bound*. The latter, whose existence is assumed, is a constant ν such that for any instant t there is an instant $t' > t$ where at all real-valued functions have their default values, and there are at most ν updates of these functions between t and t'. (Without a regular reset to default values or to values depending only on time, one cannot avoid unlimited accumulation of errors in the general case.)

[1] *Address:* Dept. of Informatics, University Paris 12, 61 Av. du Gén. de Gaulle, 94010, Créteil, France.

♯ Member of Scholars Club of Saint-Petersburg Division of Steklov Mathematical Institute, Russian Academy of Sciences

An interesting point is that the sources of desynchronization that destroy the bisimulation are much more subtle and numerous than one can think a priori. Another conceptual consequence concerns the adequacy of the notion of IA-ASM that was introduced in [2], and later studied in [3], for the specification of real-time system.

Our work in progress. We study one more question of this kind. Given a general multi-agent ASM with delayed actions, under what conditions its desynchronization gives an approximately bisimular set of runs? Practical distributed real-time controllers should be sufficiently stable, and this is one of motivations for this study.

We consider ASMs whose programs are composed of several agent ASMs that interact via shared functions. The program of each agent is constructed using updates, branching, sequential and parallel composition. Each agent has only external default loop, and agent programs are executed independently. We assume that each update is guarded (recall that we are motivated by real-time reactive controllers). The time constraints are linear inequalities as described above.

Let $\mathcal{A}$ be such ASM. By real-valued function we mean, by default, an *internal* real-valued function. There is only one external real-valued function CT. The other inputs are predicates. Let X be an occurrence of update or of guard of $\mathcal{A}$. The delays act as follows. Suppose that a run arrives at t at the evaluation of a guard or at the execution of an update, denote any of them X. This action is accomplished by an instant T that is chosen non-deterministically in the delay interval $[a_{\mathcal{A}}(X), b_{\mathcal{A}}(X)]$ attributed to X: $T \in [t + a_{\mathcal{A}}(X), t + b_{\mathcal{A}}(X)]$.

Given an ASM $\mathcal{A}_0$, its ξ-*desynchronization* $\mathcal{A}_1$ is an ASM with the same program whose delays are ξ-close to the delays of $\mathcal{A}_1$ but bigger: $0 \leq a_{\mathcal{A}_0}(X) \leq a_{\mathcal{A}_1}(X) \leq a_{\mathcal{A}_0}(X) + \xi$, $b_{\mathcal{A}_0}(X) \leq b_{\mathcal{A}_1}(X) \leq b_{\mathcal{A}_0}(X) + \xi$.

An ASM $\mathcal{A}_0$ is (ε, η)-bisimular to its ξ-desynchronization $\mathcal{A}_1$ if, for a given input, for any run ρ_0 of $\mathcal{A}_0$ there exists a run ρ_1 of $\mathcal{A}_1$ such that for each update in ρ_0 the same update in ρ_1 fires at an ε-close instant and gives an η-close value, and vice versa, from ρ_1 to ρ_0.

In order to ensure an (ε, η)-bisimilarity of $\mathcal{A}_0$ and $\mathcal{A}_1$, first, the parameter ξ of desynchronization should be smaller that ε, η divided by a constant depending on the parameters of the ASM program mentioned above, including ν, and second, the program should be (ε, η)-stable in the following sense. The internal (not containing CT) inequalities occurring in guards should be $O(\eta)$-stable, i.e., should preserve their truth value after its real-valued functions having been $O(\eta)$-perturbed (we use $\mathbb{L}^\infty$ norm). Any guard that is true at t should remains true in some time interval around t even with the values of real-valued function perturbed. Any guard that is false at an instant t should be also stably false (the exact formulations need several technical notations). At last, non identical dependent updates should be well (roughly by 2ε) separated, as well as identical self-dependent updates (like $f := f + g$). Two updates are dependent if they a have a common function in the terms that define them. We suppose that equal terms are graphically identical.

Though the constraints describing the stability are semantical, their verification for practical programs is quite feasible, and they are useful in any case as they indicate hidden traps that can destroy the presumed behavior of distributed real-time controllers.

Further research. Two main questions are of evident interest. The first question is to find sufficient 'robustness' constraints on programs that permit a *general* refinement/implementation with desynchronization. The second question concerns the verification. It is much easier to verify an abstract program specification that presumes a good synchronization. When we refine/implement the abstract program we desynchronize it, and the proven properties become false. So in order to arrive at the desired properties for the implementation/refinement we have to modify the initial requirements in order to take into account the desynchronization. In [1] we sketched a transformation that indicates what modified properties are satisfied by a concrete implementation under the condition that the initial, non modified one is satisfied by the initial abstract ideally synchronized ASM. But the question is how to modify the properties in the other direction.

References

1. J. Cohen and A. Slissenko. Implementation of sturdy real-time abstract state machines by machines with delays. Technical Report TR-LACL-2008–02, University Paris 12, Laboratory for Algorithmics, Complexity and Logic (LACL), 2008. Submitted. Available at http://www.univ-paris12.fr/lacl/.
2. Y. Gurevich and J. Huggins. The railroad crossing problem: an experiment with instantaneous actions and immediate reactions. In H. K. Buening, editor, *Lecture Notes in Computer Science, Computer Science Logics, Selected papers from CSL'95*, volume 1092, pages 266–29. Springer-Verlag, 1996.
3. D. Beauquier and A. Slissenko. A first order logic for specification of timed algorithms: Basic properties and a decidable class. *Annals of Pure and Applied Logic*, 113(1–3):13–52, 2002.

Dynamic Resource Configuration & Management for Distributed Information Fusion in Maritime Surveillance[⋆]

—Extended Abstract—

Roozbeh Farahbod and Uwe Glässer

Software Technology Lab, Simon Fraser University, Burnaby, B.C., Canada
{roozbehf,glaesser}@cs.sfu.ca

In the work described here, we devise a highly adaptive and auto-configurable, multi-layer network architecture for distributed information fusion to address large volume surveillance challenges, assuming a multitude of different sensor types on multiple mobile platforms for intelligence, surveillance and reconnaissance. Our focus is on network enabled operations to efficiently manage and improve employment of a set of mobile resources, their information fusion engines and networking capabilities under dynamically changing and essentially unpredictable conditions. Building on realistic application scenarios adopted from the design and development of the CanCoastWatch system [1–4], we contend that distributed system concepts based on decentralized control mechanisms are crucial for the design of robust and scalable network enabled operations for several reasons.

Concurrently running tasks of a distributed fusion process are created and terminated dynamically depending on mission requirements, where the workload may vary considerably due to the dynamic nature of missions. Likewise, the total number of missions to be performed concurrently varies over time depending on events that are difficult, if not impossible, to predict under all circumstances. Consequently, the overall amount and distribution of workload within the system may (and often does) change spontaneously. Even more problematic are the various external conditions that adversely affect the operational environment of the distributed fusion system. Resources allocated to tasks change dynamically as their resource capabilities vary due to transient or permanent resource failures or common events in the environment of resources, such as changing weather conditions that affect sensor ranges or communication bandwidth and range. This situation calls for reconfigurable applications that can flexibly adapt to internal changes in resource requirements as well as to external changes affecting the availability of resources. Finally, fault tolerant behavior is crucial for avoiding catastrophic system failures as a result of communication failures and partial or total resource failures.

[⋆] The work presented here has been funded under Precarn's Intelligent Systems program, NSERC Collaborative Research and Development Grant #342503-06, and MacDonald, Dettwiler and Associates.

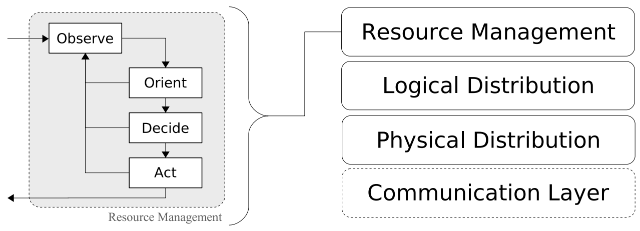


Fig. 1. DRCMA high level model

A decentralized organization of the information fusion architecture increases robustness and scalability by allowing for dynamic reorganization, thus avoiding the bottleneck of centralized fusion systems. Assuming a discrete event system model, the functional network architecture constitutes the framework for describing the events and the actions they trigger. Specifically, this includes: (1) introduction of missions, (2) decomposition of missions into tasks, (3) mapping of tasks onto the resources performing these tasks, and (4) mechanisms to actively maintain the configuration of a network of resources by observing mission requirements, resource status data, communication links, and measuring related performance values such as workload distribution and accuracy of fusion results etc. To facilitate dynamic reorganization, mobile resources are grouped into auto-configurable resource clusters with 'plug and play' features allowing for an easy migration of resources between clusters.

We present here a high-level model of our network architecture for adaptive distributed information fusion, called *Dynamic Resource Configuration & Management Architecture* (DRCMA)[5] (see also Figure 1). The DRCMA model is described in abstract functional and operational terms based on a multi-agent modeling paradigm using the *Abstract State Machine* (ASM) formalism [6] for modeling dynamic properties of distributed systems. This description of the underlying design concepts provides a concise yet precise blueprint for reasoning about key system attributes at an intuitive level of understanding, supporting requirements specification, design analysis, validation and, where appropriate, formal verification of system properties prior to actually building the system. Additionally, we also illustrate how to use the ASM formalism and underlying abstraction principles for rapid prototyping of a high-level executable DRCMA model. We do so by building on the CoreASM tool environment [7], a novel platform for experimental validation through simulation, testing and symbolic execution (model checking) of ASM models. The high level model is refined into a CoreASM model readily executable on real machines. In subsequent steps, we will extend and further refine the model into a comprehensive architecture for adaptive distributed information fusion. The result will be a prototype for testing, experimental validation and machine-assisted verification of the key system attributes prior to actually building the system. In conclusion, the proposed design approach facilitates a seamless transition across the three dimensions of modeling discrete dynamic systems: conceptual, mathematical and computational.

References

1. CCW Team: CanCoastWatch System Concept. Technical report, MacDonald, Dettwiler and Associates Ltd. (2006)
2. Wehn, H., Yates, R., Valin, P., Guitouni, A., Bossé, É., Dlugan, A., Zwick, H.: A Distributed Information Fusion Testbed for Coastal Surveillance. In: Fusion 2007. (2007)
3. Li, Z., Leung, H., Valin, P., Wehn, H.: High Level Data Fusion System for CanCoastWatch. In: Fusion 2007. (2007)
4. Farahbod, R., Glässer, U., Wehn, H.: CanCoastWatch Dynamic Configuration Manager. In: Proc. of the 14th International Abstract State Machines Workshop (ASM'07). (2007)
5. Farahbod, R., Glässer, U., Wehn, H.: Dynamic resource management for adaptive distributed information fusion in large volume surveillance. In: Proc. of SPIE Defense & Security Symposium. (2008)
6. Börger, E., Stärk, R.: Abstract State Machines: A Method for High-Level System Design and Analysis. Springer-Verlag (2003)
7. Farahbod, R., Gervasi, V., Glässer, U.: CoreASM: An extensible ASM execution engine. Fundamenta Informaticae (2007) 71–103

A First Attempt to Express KAOS Refinement Patterns with Event B

Abderrahman Matoussi, Frédéric Gervais, and Régine Laleau

LACL, Université Paris-Est
{abderrahman.matoussi,frederic.gervais,laleau}@univ-paris12.fr

1 Motivation

It is now recognised that goals play an important role in requirements engineering process, and consequently in systems development process. Whereas specifications allow us to answer the question "WHAT the system does", goals allow us to address the "WHY, WHO, WHEN" questions [1]. Up to now, the development process associated with formal methods, including Event B, begins at the specification level. Our objective is to include requirements analysis within this process, and more precisely the KAOS method. Existing work [5, 4] that combine KAOS with formal methods generate a formal specification model from a KAOS requirements model. We aim at expressing KAOS goal models with a formal language (Event B), hence staying at the same abstraction level. Thus we take advantage from the Event B method: (i) it is possible to use the method during the whole development process and (ii) we can benefit from the industrial maturity of tools supporting the method. This paper presents, through an example, the outlines of a constructive approach in which Event B models are built incrementally from KAOS goal models, driven by goal refinement patterns.

2 Goals in KAOS

KAOS (Knowledge Acquisition in autOmated Specification) [2, 3] is a methodology to implement goal-based reasoning. A goal defines an objective the system should meet, usually through the cooperation of multiple agents such as devices or humans. KAOS is composed of several sub-models related through inter-model consistency rules: (i) the central model is the *goal model* which describes the goals of the system and its environment; (ii) the *object model* defines the objects (agents ,entity...) of interest in the application domain; (iii) the *agent responsibility model* takes care of assigning goals to agents in a realisable way; (iv) the *operation model* details the operation an agent has to perform to reach the goals he is responsible for. KAOS offers a lot of refinement patterns [1] that decompose goals. These patterns can only be used in the context of different tactics defined in KAOS such as *milestone-driven tactics*, i.e. identifying milestone states that must be reached to achieve the target predicate, and *case-driven tactics*, i.e. identifying different cases to satisfy the goal. The sub-goals $G_1, G_2, ..., G_n (n \geq 2)$ refine a goal G iff the following conditions hold:

1. $G_1 \wedge G_2 \wedge \ldots \wedge G_n \models G$ (entailment)
2. For each i, j: $j \neq i$. $G_j \not\models G_i$ (minimality)
3. $G_1 \wedge G_2 \wedge \ldots \wedge G_n \not\models false$ (consistency)

In this work in progress, we focus on refinement patterns defined only with first-order logic. Patterns with LTL temporal logic will be studied in further work. Thus, the general form of the assertions associated to the patterns is $P \rightarrow Q$ where P and Q are predicates. Symbol $\rightarrow$ denotes the classical logical implication.

Let us take a simple example managing the subscription to a summer school. The main goal G states that each person who has subscribed must receive a participation receipt: $Subs \rightarrow PRcpt$. This goal is refined into three sub-goals according to the milestone-driven tactics:

(G_1) each subscription implies a payment: $Subs \rightarrow Payt$
(G_2) for each payment a bill should be issued: $Payt \rightarrow Bill$
(G_3) a receipt must be submitted whenever the bill is provided: $Bill \rightarrow PRcpt$

The case-driven tactics is applied to refine the goal G_1 into two sub-goals depending on the participant status (either student or professor):

($G_{1.1}$) $Subs \wedge Stud \rightarrow StudFee$
($G_{1.2}$) $Subs \wedge Prof \rightarrow ProfFee$

3 Expressing Goals in Event B

The objective of our work is to express a KAOS goal model with Event B. We start to study the most used refinement patterns: the milestone-driven and case-driven tactics.

Since a KAOS goal means that a property must be established, the main idea is to represent each goal as a B event and the property as the post-condition of this B event.

Thus, goal G can be translated into an abstract B event as follows:

$$\mathbf{EvG} \triangleq \text{SELECT } True \text{ THEN } Subs_B, PRcpt_B : (Subs_B \subseteq PRcpt_B) \text{ END}$$

The THEN part of **EvG** is the translation into Event B of the logical formula associated to G. Even if this translation is obviously always possible, it is not straightforward and necessarily depends on the B representation of the KAOS object model.

At this most abstract level, the guard of **EvG** is always set to *True* to express that the event is always feasible. The definitive guard is built during the refinement process.

First refinement: applying the milestone-driven tactics. All sub-goals are translated into new events using the same rules as for **EvG**. For instance, (G_1) is translated by:

$$\mathbf{EvG}_1 \triangleq \text{SELECT } True \text{ THEN } Subs_B, Payt_B : (Subs_B \subseteq Payt_B) \text{ END}$$

The abstract event **EvG** is refined by strengthening its guard. This latter is the conjunction of the post-conditions of each sub-goal:

EvG $\triangleq$
SELECT ($\mathrm{Subs}_B \subseteq Payt_B) \wedge (Payt_B \subseteq Bill_B) \wedge (Bill_B \subseteq PRcpt_B)$
THEN $Subs_B, PRcpt_B : (Subs_B \subseteq PRcpt_B)$ END

Second refinement: applying the case-driven tactics. In the same way as for the milestone-driven tactics, the sub-goals are translated by new events and $\mathbf{EvG}_1$ is refined as carried out for **EvG**.

$\mathbf{EvG}_{1.1}$ $\triangleq$
SELECT $True$
THEN $Subs_B, Stud_B, StudFee_B : ((Subs_B \cap Stud_B) \subseteq StudFee_B)$ END

However, a faithful representation of this tactics requires the following additional constraints:

– A new invariant: $(Stud_B \cup Prof_B) = Subs_B \wedge (Stud_B \cap Prof_B) = \emptyset$
– A new proof obligation: $(StudFee_B \cup ProfFee_B) \subseteq Payt_B$

Verification of the KAOS refinement conditions. Proof obligations of Event B allow most of the KAOS refinement conditions to be verified. However, for some KAOS patterns as the case-driven tactics, additional constraints must be identified.

4 Further work

The paper shows that it is possible to express KAOS goal models with Event B and furthermore, this B representation is quite close to the KAOS one. The current work is still partial and we are actively working on its extensions. Future work will be concerned with (i) generalising our method to other refinement patterns presented in [1]; (ii) considering LTL temporal operators; (iii) considering the three other sub-models of a KAOS model.

References

1. R. Darimont and A. van Lamsweerde. Formal Refinement Patterns for Goal-Driven Requirements Elaboration. In *SIGSOFT '96*, pages 179–190, San Francisco, California, USA, October 1996. ACM SIGSOFT.
2. A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *RE 2001*, pp. 249–263, Toronto, Canada, August 2001. IEEE Computer Society.
3. E. Letier. Reasoning About Agents in Goal-Oriented Requirements Engineering. Ph.D. Thesis, `ftp://ftp.info.ucl.ac.be/pub/thesis/letier.pdf`, 2001.
4. H. Nakagawa and K. Taguchi and S. Honiden. Formal specification generator for KAOS. In *ASE 2007*, pages 531–532, Atlanta, Georgia, USA, November 2007. ACM.
5. C. Ponsard and E. Dieul. From Requirements Models to Formal Specifications in B. In *REMO2V'2006*, Luxembourg, June 2006.

Verification and Validation of Web Service Composition Using Event B Method

Idir Ait-Sadoune and Yamine Ait-Ameur

LISI/ENSMA - Université de Poitiers
Téléport 2 - 1, avenue Clément Ader - B.P. 40109.
86960 Futuroscope Cedex - France.
{idir.aitsadoune,yamine}@ensma.fr

The Service-Oriented Architecture based on the Web service technology emerged as a consequence of the evolution of distributed computing. A web service is a set of software operations offered on the web by a provider. It is described in a WSDL[1] language and published in a UDDI[2] directory so that it can be found and invoked by a user. One of the key ideas of this technology is the ability to create service compositions by combining and interacting with pre-exisiting services to offer a more complex functionality. *Orchestration* is the process that allows to schedule the defined services compositions and BPEL[3] is the most known and used orchestration language. BPEL allows the designer to represent service compositions by various behavioral properties like services interactions (message exchanges), control flow constraints (sequence, iteration, conditional) or data flow constraints (exchange, modification, evaluation of data expressions).

The resulting process from the compositions of independent services must obey to some behavioral requirements in order to achieve its functional goal. These requirements include deadlock freeness in the composition execution, correct manipulation and transformation of data, satisfying rules and constraints on ordering between interactions and termination. The web services compositions languages, like BPEL, and the operational orchestration tools associated with these languages like orchestra[4], support the encoding and the interpretation of the composition process. But they don't support expression and verification of behavioral requirements.

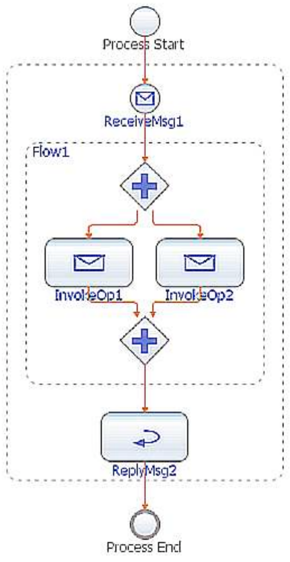
Various approaches have been proposed to model and to analyze web services and services composition. We quote the work realized using Process Algebra[5], Petri Nets[6] and Timed properties[7]. The major contributions of this work is based on the model checking technique.

We propose to address the problem of services composition validation using proof and refinement based techniques, in particular the event B method. Our approach consists in extracting an event B model from service compositions described in BPEL. Thereafter, the obtained model is enriched with the relevant properties in the INVARIANTS and THEOREMS clauses and events guards. The consistency checking of the resulting model is established using the RODIN platform[8] and animation is performed with the B2EXPRESS tool[9].

In this paper we present a brief description of the BPEL language and the approach we have proposed.

1 BPEL

BPEL (Business Process Execution Language[3]) is a standard for specifying and executing business processes. The process definition consists of several parts describing partner links, process variables, main process workflow and other parts which are not defined in this paper. The partner link declarations are used to define the relation between the process and its partners. The process variables are used to represent the state of the business process. The process flow is defined by a set of process activities. It specifies the operations to be performed, their ordering, activation conditions, reactive rules, etc. Basic activities represent primitive operations performed by the process (*invoke*, *receive*, *reply*, *assign*, *terminate*, *wait* and *empty* activities). The structured activities include *sequence*, *switch*, and *while* that model traditional control constructs, *pick* for a nondeterministic choice based on external events (i.e., message reception or timeout), *flow* activity for parallel execution of nested activities. Figure 1 shows an example of a BPEL description. It defines a *sequence* of three activities (*RecieveMsg1*, *Flow1* and *ReplayMsg2*). *Flow1* is decomposed into two *parallel* operations (*InvoqueOp1* and *InvoqueOp2*). Figure 1.a shows a graphical representation like it is designed in the NetBeans[10] tool and figure 1.b shows the corresponding and generated XML representation.



```
 <variable name=Msg1 messageType=t1 .../>
 <variable name=Msg2 messageType=t2 .../>

 <sequence>
   <receive name="ReceiveMsg1"
            operation="Oper1"
            variable="Msg1"
            .../>
   <flow name="Flow1">
      <invoke name="InvokeOp1"
              operation="Oper1"
              InputVariable="Msg1"
              OutputVariable="Msg2"
              .../>
      <invoke name="InvokeOp2"
              .../>
   </flow>
   <reply name="ReplyMsg2"
              .../>
</sequence>
```

Fig. 1. (a) Graphical representation of BPEL. (b) XML representation of BPEL

2 From BPEL to event B

The proposed formal modeling of BPEL by event B starts from the observation that a BPEL definition is interpreted as a transition system. A state is represented in both languages by a *variables* element in BPEL and by the VARIABLES clause in event B. The various activities of BPEL represent the transitions, they are represented by the events of the EVENTS clause in the B language.

The modeling process is inductively defined on the structure of the BPEL definition. It is based on the following rules :

1 - each variable in the BPEL language corresponds to a state variable of the event B model in the VARIABLES clause (see table 1);

2 - each activity becomes an event of the B model;

3 - each composition operation of BPEL activities (*flow*, *sequence*, *foreach*, *if then else...*) becomes a construction of event B. This representation is based on the translation rules defined to encode process algebra expressions in event B models of [11]. It uses the refinement to represent the temporal decomposition of the activities.

The obtained model is enriched by the expression of the various properties to be checked defined in INVARIANTS and THEOREMS clauses of the event B model. For example, INVARIANTS, THEOREMS and VARIANT clauses allow a developer to express deadlock and live lock freeness.

`<variables>`
`    <variable name="Msg1" messageType="t1"/>`
`    <variable name="Msg2" messageType="t2"/>`
`...`
`</variables>`
	`VARIABLES`
	`    Msg1, Msg2`
	`INVARIANTS`
	`    Msg1 ∈ t1 ∧`
	`    Msg2 ∈ t2`

Table 1. Example of modeling of BPEL Variables elements by event B VARIABLES clause

1. They shall express that the new events of the concrete model are not fired infinitely (no live lock). *A decreasing variant is introduced for this purpose.*
2. They shall express that, at any time, an event can be fired (no deadlock). *This property is ensured by asserting (in the THEOREMS clause) that the disjunction of all the abstract events guards implies the disjunction of all the concrete events guards.*

`...`
`<invoke name=invokeQueryOperation`
`    Operation=QueryOperation`
`    InputVariable=Msg1`
`    OutpuVariable=Msg2`
`    PartenerLink=queryPL/>`
	`EVENTS`
	`...`
	`invokeQueryEvent =`
	`    BEGIN`
	`        Msg2 := QueryOperation(Msg1)`
	`    END`

Table 2. Example of modeling of BPEL activity by an event of B model

3 The validation methodology

The validation of services composition expressed in BPEL by event B models can be performed according to two scenarios.

1 - A BPEL description is fully translated into a single event B model (no intermediate refinement) then it is validated. At this level, we may be faced to complex proof obligations.

2 - Each decomposition of a complex activity in BPEL is translated into event B by refining the event corresponding to this activity. This refinement introduces the temporal decomposition defined in the original BPEL specification. A step by step BPEL description and validation is performed in parallel.

As a conclusion, event B models can be used to check the consistency of BPEL specifications.

References

1. W3C: Web Services Description Language (2007) http://www.w3.org/TR/wsdl.
2. OASIS: Universal Description, Discovery, and Integration Specification (2003) http://uddi.xml.org/.
3. OASIS: Business Process Execution Language (2007) http://bpel.xml.org/.
4. BSOA: Orchestra (2006) http://orchestra.objectweb.org.
5. Foster, H.: A Rigorous Approach To Engineering Web Service Compositions. PhD thesis, Imperial College London, University Of London (January 2006)
6. Tang, Y., Chen, L., He, K.T., Jing, N.: An Extended Petri-Net-Based Workflow Model for Web Service Composition. In: ICWS. (2004) 591–599
7. Kazhamiakin, R.: Formal Analysis of Web Service Compositions. PhD thesis, Università degli Studi di Trento (March 2007)
8. ClearSy: Rodin (2006) http://www.clearsy.com/rodin/industry_day.html.
9. Ait-Sadoune, I., Ait-Ameur, Y.: B2EXPRESS, Un animateur de modèles B événementiel. In: AFADL, Namur, Belgique (13-15 Juin 2007)
10. NetBeans: SOA Application Learning Trail (2006) http://www.netbeans.org/kb/trails/soa.html.
11. Aït-Ameur, Y., Baron, M., Nadjet, K.: Encoding a Process Algebra Using the Event B Method. Application to the Validation of User Interfaces

XML Database Transformations with Tree Updates

Qing Wang[1], Klaus-Dieter Schewe[2] and Bernhard Thalheim[3]

[1] Massey University, New Zealand
q.q.wang@massey.ac.nz
[2] Information Science Research Centre, Palmerston North, New Zealand
k-d.schewe@xtra.co.nz
[3] Institute of Computer Science, CAU Kiel,Olshausenstr. 40,Kiel, Germany
thalheim@is.informatik.uni-kiel.de

1 Introduction

For many years the eXtensible Markup Language (XML) has attracted much research attention from database communities, particularly in the area of query and transformation languages such as XQuery and XSLT. XML documents are usually represented as trees. In order to accommodate the diversity of user requirements, it is desirable to conduct transformations on XML trees at flexible abstraction levels. However, most of current approaches have a fixed abstraction level at which updates must be identified for individual nodes and edges. In this paper we investigate XML database transformations with structured updates, for example, manipulations on portions of a tree, including deleting, modifying or inserting subtrees, copying contexts, etc. To accomplish this task, Abstract State Machines (ASMs) will be employed as it has turned out in [3] to be a universal computation model capturing database transformations.

As in the study of algorithms [1], the problem of partial updates will come up again. In essence, partial updates root in two factors: complex objects and parallel computing. When several parallel computations are executing updates on partial parts of the same complex object, inconsistency of an update set might arise. As trees are typically a kind of complex objects, we believe that XML database transformations provide an interesting paradigm for the study towards partial updates.

2 Tree Algebra, Tree Updates and ASMs

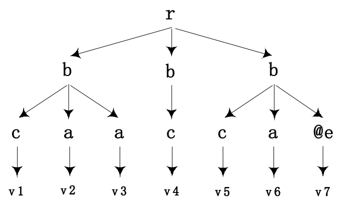
XML Tree Model. We consider an XML document as an *unranked tree* in which the number of the children of a node may be unbounded. As shown in the tree (i) of document exa.xml, elements, attributes and character data of an XML document correspond to nodes of an XML tree. Formally, let N^* denote the set of finite strings over positive integers, A be a set of labels, and Σ be an alphabet called leaf alphabet, then an XML tree t is a pair ($Dom(t)$, λ_t) for a finite, prefix-closed set $Dom(t) \subseteq N^*$ and a mapping λ_t: $Dom(t) \rightarrow A \cup \Sigma^*$, satisfying the following conditions:

– if $\lambda_t(n) \in A$ and node n has k children, then $\{i|ni \in Dom(t)\} = \{1, ..., k\}$;
– if $\lambda_t(n) \in \Sigma^*$, then $\{i|ni \in Dom(t)\} = \emptyset$.

A tree t_1 is said to be the *subtree* of a tree t_2 at node n iff the following properties are satisfied: (i) $Dom(t_1) = \{i|ni \in Dom(t_2)\}$ and (ii) $\forall n^{'} \in Dom(t_1), \lambda_{t_1}(n^{'}) = \lambda_{t_2}(nn^{'})$. For convenience, we use $\widehat{n}$ to denote the subtree of t rooted at n for $n \in Dom(t)$. A sequence $[t_1, ..., t_k]$ of trees is called a *hedge* and a set $\{t_1, ..., t_k\}$ of trees is called a *forest*. Note that, we consider forests as a special kind of hedges in which the order of trees is not significant, for example, a collection of trees yielded by parallel computations is a forest. An XML database is an unordered collection of XML trees.

⟨!ELEMENT r(b*)⟩
⟨!ELEMENT b(c, a+)⟩
⟨!ATTLIST b e CDATA #IMPLIED⟩
⟨!ELEMENT c(#PCDATE)⟩
⟨!ELEMENT a(#PCDATA)⟩

exa.xml



(i)

Tree Algebra. We extend the algebras for trees introduced by [4, 2] to the case of XML trees. The signature of an XML tree consists of three sorts: $\mathcal{L}$ (for labels), $\mathcal{H}$ (for hedges), $\mathcal{C}$ (for contexts) and a set $\mathcal{F}$=$\{\iota, \delta, \zeta, \rho, \kappa, \eta, \sigma\}$ of function symbols such that $\iota : \mathcal{L} \times \mathcal{H} \to \mathcal{H}$, $\delta : \mathcal{L} \times \mathcal{C} \to \mathcal{C}$, $\zeta : \mathcal{H} \times \mathcal{C} \to \mathcal{C}$, $\rho : \mathcal{H} \times \mathcal{C} \to \mathcal{C}$, $\kappa : \mathcal{H} \times \mathcal{H} \to \mathcal{H}$, $\eta : \mathcal{C} \times \mathcal{H} \to \mathcal{H}$ and $\sigma : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$. The set $\mathcal{T}$ of terms over $A \cup \Sigma \cup \{\varepsilon, \xi\}$ is constituted by terms over three sorts, i.e., $\mathcal{T} = T_{\mathcal{L}} \cup T_{\mathcal{H}} \cup T_{\mathcal{C}}$, where $T_{\mathcal{L}}$, $T_{\mathcal{H}}$ and $T_{\mathcal{C}}$ stand for label, hedge and context terms, respectively.

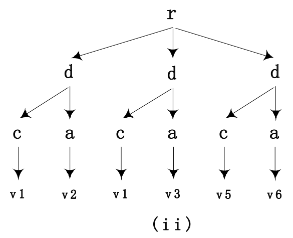
– $T_{\mathcal{L}}$= $A \cup \Sigma^*$.
– $T_{\mathcal{H}}$= $T^s_{\mathcal{H}} \cup T^m_{\mathcal{H}}$ such that
  - $\varepsilon \in T^s_{\mathcal{H}}$,
  - $\ell_t \in T^s_{\mathcal{H}}$ for $t \in \Sigma^*$,
  - $t\langle t_1, ..., t_n\rangle \in T^s_{\mathcal{H}}$ for $t \in A$ and $t_1, ..., t_n \in T^s_{\mathcal{H}}$, and
  - $[t_1, ..., t_n] \in T^m_{\mathcal{H}}$ for $t_i \in T^s_{\mathcal{H}}$ $(i = 1, ..., n)$.
– $T_{\mathcal{C}}$ is the smallest set defined by:
  - $t\langle t_1, ..., t_n\rangle \in T_{\mathcal{C}}$ for $t \in A$ and $t_1, ..., t_n \in T^s_{\mathcal{H}} \cup \{\xi\}$ such that exactly one $t^{'} \in \{t_i | i = 1, ..., n\}$ is ξ.

The extended tree algebra Λ is a pair $(\mathcal{T},\mathcal{F})$ satisfying certain axioms. By this algebraical approach, XML trees with their features, such as ordered elements, unbounded numbers of children for nodes, etc., can be constructed.

Tree Updates with ASMs. Now we give several examples to illustrate how ASMs can be used as a computation model for XML database transformations. The navigation part of transformations is taken from XPath. Besides the usual rules of ASMs an additional update rule $t_1 += t_2$ is used to express that a tree t_2 is added into the forest t_1. Other definitions that are irrelevant to the following examples are skipped.

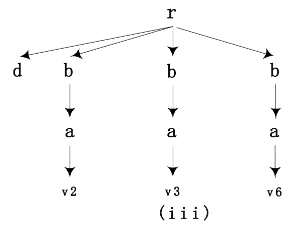
Example 1. The tree (ii) is constructed from subtrees of the tree (i).

```
forall $x$,$y$,$z$ with $x \in doc(\text{“}exa.xml\text{”})/r/b$
                  $\wedge y \in x/c \wedge z \in x/a$
do
    $t_1 := \iota(d, \kappa(\widehat{y}, \widehat{z}))$;
    $t_2 \mathrel{+}= t_1$
enddo;
$output := \iota(r, t_2)$
```



Example 2. The tree (iii) is constructed by applying subtrees of the tree (i) into a specified context.

```
forall $x$,$y$ with $x \in doc(\text{“}exa.xml\text{”})/r/b$
                  $\wedge y \in x/a$
do
    $t_1 := \iota(b, \widehat{y})$;
    $t_2 \mathrel{+}= t_1$
enddo;
$output := \eta(\rho(t_2, \delta(r, \xi)), \iota(d, \varepsilon))$
```



After further adding tree predicates as locations for subtrees and contexts of an XML tree, existing XML trees can be updated by replacing identified subtrees and contexts with others of the same sorts. Consequently, the consistency checking on a collection of updates produced within a computation step becomes important.

3 Discussions and Future Work

To minimize the inconsistency caused by partial updates, one of approaches is to check the compatibility of updates whose locations are overlapping. This can be conducted both at the schema level and at the instance level. Moreover, such a computation model for XML database transformations motivates us to seek for links between ASMs and tree transducers. We will exploit them in the future.

References

1. Gurevich, Y., and Tillmann, N. Partial updates. *Theor. Comput. Sci. 336*, 2-3 (2005), 311–342.
2. Walukiewicz, I., and Bojanczyk, M. Forest algebras. In *Logic and Automata* (2007), J. Flum, E. Graedel, and T. Wilke, Eds., Amsterdam University Press.
3. Wang, Q., and Schewe, K.-D. Axiomatization of database transformations. In *Proceedings of the ASM'07: The 14th International ASM Workshop* (2007).
4. Wilke, T. An algebraic characterization of frontier testable tree languages. *Theor. Comput. Sci. 154*, 1 (1996), 85–106.

Weaving Authentication, Authorization and Auditing Requirements into the Functional Model of a System using Z Promotion

Ali Nasrat Haidar and Ali E. Abdallah

E-Security Research Centre
London South Bank University
103 Borough Road
London SE1 0AA, UK
`{ali.haidar,a.abdallah}@lsbu.ac.uk`

Abstract. The use of Z in software development has focused on specifying the functionality of a system. However, when developing secure system, it is important to address fundamental security aspects, such as authentication, authorization, and auditing. In this paper, we show an approach for building systems from generic and modular security components using promotion technique in Z. The approach focuses on weaving security component into the functionality of a system using *promotion* technique in Z. For each component, *Z* notation is used to construct its state-based model and the relevant operations. Once a component is introduced, the defined local operations are promoted to work on the global state. We illustrate this approach on the development of a "secure" model for a conference management system. With this approach, it is possible to specify the core functionalities of a system independently from the security mechanisms. Authentication, authorization, and auditing are viewed as components which are carefully integrated with the functional system.

Key words: Z specification, Security Requirements, Authentication, Authorization, Auditing, Weaving Security into Functional Models, Z Promotion

Separation of Z operations

Ramsay Taylor

Dept.of Computer Science, Regent Court, University of Sheffield, S1 4DP, UK
ramsay@dcs.shef.ac.uk

1 Introduction

Machine code and assembly language programs are structured using various branches and decision points, but between these they contain blocks of instructions that are simply sequentially composed. Most work on formal program analysis has focused on the behavior of the branch points — primarily because composing the blocks of sequential code to determine their overal effect on the system is often intellectually trivial. This processs is also computationaly simple, but it is not computationally trivial. The sequential blocks can comprise a large proportion of the instructions. If we want to apply formal reasoning to the extremely large sets of instructions that result from analysing code in low-level languages we will want to automate the process. It would then be useful to be able to make the intellectually trivial parts of the analysis computationally trivial as far as possible. The aim of this work is to produce a system of rules that can be efficiently implemented[1] and allow us to determine the overal behaviour of sequentially composed operations.

To identify those sequential compositions that are trivial we will use techniques inspired by Separation logic[3, 1]. Separation logic itself is a very general, abstract collection of higher order logic statements that covers a huge semantic range. To apply separation logic to our context would require the use of serious theorem proving and so would not reduce the computational burden. However, the simple observation at the heart of separation logic can be used: if two operations refer to completely disjoint parts of the state space they can be reasoned about independently.

Here we will not present anything with the generality and elegance of separation logic. Nor will we present a complete solution to analysing sequential composition in Z. The aim is to present some techniques that are very easy to implement and that will identify those operation compositions that are trivial. These can be processed syntactically, before a more serious theorem proover is applied.

The approach taken is in the spirit of separation logic. If two operations are sequentially composed but it can be shown that the effects of the first in no way influence the effects of the second then the effect of the composition is just the syntactic combination of the two — a new schema for the composition can be

[1] This is acheived primarily by operating at a textual level with almost no parsing and semantic processing.

created by concatenating all the declarations and predicates together after some very simple pruning.

2 Separation in Z

In order to determine the overall effect of a sequence it must first be determined how the individual effects of the operations interact. The simplest but most crucial question is whether they interact at all. Where an operation is unaffected by the preceding operation we shall say that they are *Separate* and shall use the symbol '&' after Reynolds [2].

For any Z operation we can require that every state variable referenced is brought into scope explicitly. We can then collect these variables together into two state schemas that we shall call $State_{\Delta}$ and $State_{\Xi}$. The second of these, $State_{\Xi}$, will contain all the variables that are unchanged by the operation. The first schema, $State_{\Delta}$, will contain all the others - i.e. all the variables that are changed or whose resulting state is left unspecified.

We can now write the most general definition of our version of separation. The notation we shall use is $Op_1 \& Op_2$ if, in the sequential composition $Op_1 \fatsemi Op_2$, the effect of Op_2 is unaffected by the actions of Op_1. For example: $P == [x, x', y, y' \mid x' = x + y \land y' = y]$ and $Q == [y, y' \mid y' = y + 1]$. Here P doesn't alter y, so does not alter the effect of Q on the system state[2]. With some subtlety to protect against referencing of variables, even if they aren't changed, our definition for separation becomes:

$$Op_1.State_{\Delta} \cap (Op_2.State_{\Delta} \cup Op_2.State_{\Xi}) = \varnothing \ \langle \texttt{Total Separation} \rangle$$

This also serves as an example of the difference between separation and commutativity. In our study of separation here we are not interested in showing that two operations *never* interfere with each other, simply that in the presented usage their effects on the state of the system are independent.

Although the `Total Separation` definition presented above does satisfy our requirements for identifying independent operations it is too restrictive in some cases. For example, an operation R that changes the value of x and an operation S that brings x into scope but simply leaves it unchanged and does not use its value to determine anything else[3].

This is clearly an over-restriction, since S doesn't use the value of x in determining any of its effects. Also, since S leaves x unchanged it doesn't affect our other requirement: if we want to consider the overall effect of the composition we need to know that any effects produced by R aren't undone by S. With `Total Separation` this was implicit, since the second operation wasn't allowed to even *mention* anything changed by the first. When relaxing the rule to allow situations such as $R \& S$ we must be careful not to violate this requirement.

[2] This is also a good example of the difference between our notion of non-interference and comutativity — $Q \& P$ is *not* true.

[3] This may seem pointless but it occurs often in Z specifications — generally where a state schema is brought into scope but only some of its variables are needed.

We need to specify that one of the operations should *only* refer to the variable in the proposition that leaves it unchanged. So, if both operations refer to x one of them should only do so to state that it is unchanged; that is it should contain $[x' = x]$ and no other propositions that refer to x. The neatest way to identify this property is to say that, if we removed the statement $[x' = x]$ from one of the operations it would no longer refer to x in any way. This gives us the following new definition for separation:

$$\begin{aligned} &\forall\, x \mid x \in (Op_1.State_\Delta \cap (Op_2.State_\Delta \cup Op_2.State_\Xi)) \bullet \\ &\quad \exists\, Op_{2a} \mid Op_2 == Op_{2a} \wedge [\Xi x] \bullet Op_1 \& Op_{2a} \\ &\qquad\qquad\qquad\qquad\qquad \langle\texttt{Effective Separation}\rangle \end{aligned}$$

Clearly the separation requirement for $Op_1 \& Op_{2a}$ can be satisfied either by the `Total Separation` definition or by recursive application of this definition if there are multiple variables to be considered.

This can be further extended to cope with operations that modify Z functions, but that modify disjoint sections of the domain.

3 Automation

A prototype implementation of the ideas presented here has been produced. It is written in Java and there is no formal demonstration of correctness, but it does show the ease of implementation that was intended for this work and the speed of execution.

The following results were produced with randomly generated Z operation schema[4] on a 1.8GHz Pentium 4[5].

```
Completed 10100 comparisons in 1299ms (0h 0m 1s)
Completed 250500 comparisons in 26473ms (0h 0m 26s)
Completed 1001000 comparisons in 96919ms (0h 1m 36s)
Completed 10243200 comparisons in 1000160ms (0h 16m 40s)
```

Bibliography

[1] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *proceedings of POPL'01*, 2001.

[2] J. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of LICS 2002*, pages 55–74, 2002.

[3] J. C. Reynolds. Intuitionistic reasoning about shared mutable data structure. In *Proceedings of the Symposium in Celebration of the Work of C.A.R. Hoare*, 1999.

[4] The random generator is fairly simplistic. It currently never generates schema with functional types.

[5] These results are purely indicative; the tests were run in the background while various other applications were in use.

Object modelling in the SystemB industrial project

Helen Treharne, Edward Turner, Steve Schneider [1], Neil Evans[2]

[1]Department of Computing, University of Surrey
[2]AWE plc Aldermaston

Abstract. The SystemB project is a two year project at the University of Surrey, funded by AWE plc, and is concerned with bridging the areas of formal methods and object modelling. The project is focused on the CSP ‖ B integrated formal method and increasing its level of tool support so that CSP ‖ B models of Executable UML (xUML) systems can be constructed automatically. The CSP ‖ B models will subject the xUML model to formal analysis prior to generating executable code. We are currently developing a CSP ‖ B model generator within the xUML toolsuite provided by Kennedy Carter Ltd. xUML is used within AWE and we will initially focus on reasoning about xUML state machines. Actions within xUML state machines are defined using the Action Specification Language (ASL). ASL is more low level than the Object Constraint Language; they can execute concurrently, and can also be used in operation definitions. Hence it is a challenge to model formally. In this extended abstract we provide an overview of one ASL to AMN translation pattern being developed and highlight the role of B in the project.

1 Introduction

The University of Surrey and AWE plc are collaborating on a two year project that aims to support the definition of UML models with formal analysis. The motivation is to increase engineers' confidence in the UML models of a system early in the software development life cycle. The application domain of interest is safety critical and therefore it is essential to achieve a high level of assurance in safety of the models, i.e., that they preserve desirable behavioural properties and are deadlock-free. The UML models are executable and are currently validated by running numerous simulations. Our aim is to automatically generate CSP ‖ B [6] models, corresponding to such UML models, which can be formally analysed using FDR [1] and ProB [3]. The challenge is to develop a translation with tool support so that the effort is spent on conducting the formal analysis rather than building formal models. Another tool that produces a formal model from a UML-like notation is UML-B [7], where the emphasis is on providing a graphical interface to Event-B rather than analysing the integrity of a UML model.

The project will consider two different routes for developing a *CSP ‖ B model generator*. Firstly, we will develop a model generator using the Executable UML [5] toolsuite provided by Kennedy Carter Ltd. This offers the capability of code generation into C, C++ or Java from platform independent models, and it is used as a technology within AWE. AWE have been working alongside Kennedy Carter to develop Spark Ada translators from xUML. Thus, our SystemB tool will enable our formal analysis support to fit into the AWE software development life cycle. Secondly, we will also build a model generator using the Epsilon [2] toolset being developed at York University. This will involve generating CSP || B models from UML models using model-text transformations based on the Epsilon Generation Language. It will echo the work done in the first model generator but will enable us to gain experience in developing tool support based on an Eclipse plug-in. We will then move on to developing CSP || B meta-models so that a more general and extensible model generator can be developed.

The current focus of the project is on translating xUML state charts and class diagrams. State charts specify the behaviour of instances of a class, and each class has at most one state chart. Each state comprises a single entry action, defined using the Action Specification Language (ASL). The current translation follows the CSP || B pattern that separates the specifications for control flow and data. Accordingly, object lifetimes are specified using CSP, which are based on state charts in UML; system data, including object instance handles and attribute information, is stored within B machines. Attributes are expressed as functions over instance handles, and an additional machine is generated to specify the relationships used in the model, such as associations and generalisations.

2 Overview of ASL

ASL is more low level than the Object Constraint Language (OCL) [4]. OCL statements have no side effects and execute immediately. Conversely, ASL statements can change the state of a system, and can be grouped into blocks, e.g., within operation methods, or the states of a state chart. Since actions of a state in a state chart actions can execute concurrently, so may ASL statements.

The next section describes the translation of a frequently used ASL statement to show the type of translation our tool must perform.

ASL Translation Example: Consider a *Person* class with two attributes, *age* and *glasses*, and an associated state chart. The SystemB tool translates this system to a CSP specification that specifies the lifetime of a *Person* object, and a B machine storing *Person* object data containing:

– a set called, *PERSON*, denoting all objects of this class,
– a variable, $personIH \subseteq PERSON$, recording the *PERSON* objects that have been created, but have not yet been deleted (the active instances),
– a variable, $age \subseteq personIH \rightarrow \mathbb{N}$, recording the age of the active instances,
– a variable, $glasses \subseteq personIH \rightarrow Bool$, denoting whether the active instances require glasses to correct their vision.

The ASL `find` statement identifies a set of object instance handles satisfying a certain condition. For example, the following statement finds the active *Per-*

son instances, older than 60, who require glasses to correct their vision (where `PersonSet` is a subset of *personIH* that has been instantiated previously):

```
{res} = find PersonSET where age > 60 & glasses = TRUE
```

Initial efforts towards translating such statements included using constant lambda functions to encode the statement conditions. This is made difficult when using ProB for analysis since their instantiation can be very time consuming for large systems. Translating `find` in general is not straightforward because it requires evaluating arbitrary predicates in a static environment. The current approach generates a tailored B operation for each `find` statement of the same form. The translated B operation, given below, accepts parameters corresponding to the statement variables (`PersonSet`, `age` and `glasses`), whose action uses a set comprehension to identify the instances that satisfy the condition:

```
$res \leftarrow$ **findExample**( $setih$, $a$, $g$ ) $\widehat{=}$
    **PRE**    $setih \subseteq \text{PERSON} \wedge a \in \text{ran}(age) \wedge g \in \text{ran}(glasses)$
    **THEN**   $res := \{s \mid s \in setih \wedge age(s) > 60 \wedge glasses(s) = g\}$
    **END**
```

3 Discussion

The above statement directly maps to one B operation and this style is also applicable to other ASL statements concerned with object management, e.g., creation/deletion. ASL statements which support transitions between states typically translate into CSP events. Our tool automatically generates CSP $\|$ B models that handle these ASL statements. The difficult ASL statements to translate are programming language constructs, e.g., loops. These are not easily expressed using B or CSP alone, but may require a mixture of both, and in fact we need to reconsider whether we should be translating at this low level. The challenge is that we must capture the essence of what the ASL is describing so that the formal model contains just enough information to conduct a meaningful analysis.

References

1. Formal Systems Oxford: *FDR 2.83*. http://www.fsel.com, 2007.
2. Kolovos, D.S., Paige, R.F., and Polack, F.A.C.:*Epsilon Development Tools for Eclipse*, Proc. Eclipse Summit 2006, Esslingen, Germany, October 2006.
3. Leuschel, M. and Butler, M.:*ProB: A Model Checker for B*, FME Symposium, 2003.
4. Object Management Group, ptc/03-10-14: *UML 2.0 OCL Specification*, 2003.
5. Raistrick, C., Francis, P., Wright, J., Carter, C., Wilkie, I.: *Model Driven Architecture with Executable UML.* Cambridge University Press, 2004.
6. Schneider, S., Treharne, H.:*CSP Theorems for Communicating B machines*, FACS 17(4) 2004.
7. Snook, C. and Butler, M.: *UML-B and Event-B: an integration of languages and tools.* Proc. Software Engineering 2008, Innsbruck, Austria, February, 2008.
8. Wilkie, I., King, A., Clarke, M., Weaver, C., Raistrick, C. and Francis, P.: *UML ASL Reference Guide (ASL language level 2.5)*, Kennedy Carter Ltd., 2003.

FDIR Architectures for Autonomous Spacecraft: Specification and Assessment with Event-B

Jean-Charles Chaudemar[1], Charles Castel[2], and Christel Seguin[2]

[1] ISAE-DMIA, Toulouse, France
[2] ONERA-DCSD, Toulouse, France

Abstract. On-board Fault Detection, Isolation and Recovery (FDIR) systems are considered to ensure the safety and to increase the autonomy of spacecrafts. They shall be carefully designed and validated. Their implementation involves a relevant knowledge of items like functions and architectures of the system, and a fault model in relation with these items. Thus, the event-B method is well suited to correctly specify and validate on-board safety architectures.
This paper focuses on the FDIR concept presentation and the use of event-B for formalising and for refining the FDIR concept.

1 Introduction

On-board Fault Detection, Isolation and Recovery (FDIR) systems aim at maintaining the safe spacecraft operation even when faults occur. They also enable to limit service interruptions with reduced ground operations. So, they contribute to the spacecraft autonomy.

They are complex systems composed of various mechanisms, ranging from the monitoring and activation of basic physical devices to the reconfiguration of the spacecraft mission. They are often structured in layers to master this complexity. One issue is to characterize and validate each layer and the relationship between each layer.

These characteristics require a rigorous and progressive validation of the system from the early phases of design by discharging proof obligations. Accordingly, it is necessary to use a formal method like event-B for the modelling and proof.

This paper focuses on the FDIR concept presentation and the use of event-B for formalising and for refining the FDIR concept.

The paper is organised as follows: after a short presentation of on-board FDIR concept strongly bounded with autonomy architecture concept, the next section suggest activities enabling to implement FDIR concept. Then, we present the framework of formal modelling that we will use to describe our architecture and the properties related to this architecture. The last section deals with the objectives for the future work.

2 FDIR concept

FDIR is the means to detect off-nominal conditions, isolate the problem to a specific subsystem/component, and recover of vehicle systems and capabilities [1]. In this paper, FDIR is considered as an operational function that contributes to the autonomy of the system and whose main purpose is to maintain the availability and the safety of the system [2].

The main activities related to FDIR design concern:

- identification of fault-classes that could impact the requested availability and safety objectives;
- proposition of a strategy in order to tolerate above fault-classes. A strategy suggests a logical solution to achieve these objectives;
- implementation of this strategy by proposing a static and dynamic architecture: combining functional components with safety components for diagnosis and reconfiguration.

One difficulty is often the lack of traceability between these activities. The strategy that gave the rationales of the architecture is left implicit. The proposed architecture is often the result of expert judgement. So it is hard to prove the consistency between the objectives and the architecture. Therefore our proposition is to model the objectives of the concerned system using event-B method. We first model some strategies or patterns of safety that allow to meet requirements described in the objectives of the system. Then, we show how these patterns are refined rigorously in concrete architectures by discharging proof obligations.

3 Work in progress using event-B method

Event-B is a formal method for the development of complex system. Its formalism supports the validation of some properties thanks to proof methods. Thanks to these distinctive features, the event-B method is well suited to correctly specify and progressively validate on-board safety architectures.

Let us illustrate how the three activities are modelled in event-B. For mission objectives of a spacecraft, two feared events are identified: the spacecraft loss and the interruption the mission. Safety architecture patterns propose micro architecture solutions that enable to mitigate such feared events. We propose to reuse and extend the patterns presented in [3].

For this paper, we model more specifically a safety architecture pattern that includes a primary functional component and a redundant one, under the hypothesis of no common fault. The safety property to be met is: "one single fault shall not lead to the total loss of the function". We modelled this pattern at three abstraction levels successively refined which verify this property.

The most abstract model enables to formalise this property and hypothesis. Two basic components are considered. A fault counter (*fault_ci*, with i stands for 1 or 2) is associated to each component, whereas a boolean status (*status*) models the global system health. When a fault occurs (event *disci*, with i stands for 1 or 2), the component "i" is considered disconnected. When the event *final* occurs, nothing more happens, since the system remains in the faulty state. Moreover, for all this behaviour, the safety property is expressed by an invariant: $card(fault\_c1) + card(fault\_c2) \leq 1 \Leftrightarrow status = TRUE$. But at this step, there is no detail about the condition of spare component activation.

Accordingly, the second model refines the former with introduction of the switching process as a strategy. The activation variables are set in this new model. Two new events are defined: *sw1_2* event allows to switch from the primary active component to the spare component which becomes active when the primary one is disengaged; *sw2* event disconnects the system by disconnecting the secondary component.

At least, the third model is an implementation of this switching strategy by taking into account data flow: *normal1* (respectively *normal2*) event represents the "normal" data flow of the primary (respectively, the spare) component according to the specification; *fail1* (respectively *fail2*) event represents the "faulty" data flow of the primary (respectively, the spare) component.

4 Future work

The current work investigates B-event specifications and refinements of generic FDIR strategies by using the RODIN platform. The future work will consist in studying the impact of the concept of a "layered" FDIR and how it can be modelled and validated in the B-event framework, in the same spirit than the full constructive approach developed by [4].

References

1. NASA: Glossary - NASA Crew Exploration Vehicle, SOL NNT05AA01J, Attachment J-6. `http://www.spaceref.com/news/viewsr.html?pid=15201` (2005)
2. Chaudemar, J.C., Castel, C., Gabard, J.F., Tessier, C.: Z and ProCoSA based specification of a distributed FDIR in a satellite formation. In: CAR'07 - Second National Workshop on Control Architectures of Robots, Paris, FR (2007)
3. Kehren, C.: Motifs formels d'architectures de systèmes pour la sûreté de fonctionnement. SUPAERO, Toulouse, FR. (2005)
4. Arora, A., Kulkarni, S.: Component based design of multitolerant systems. Software Engineering, IEEE Transactions on **24**(1) (1998) 63–78

Using ASM to achieve executability within a family of DSL

Ileana Ober, Ali Abou Dib

IRIT – Université Paul Sabatier Toulouse
118, route de Narbonne 31062 Toulouse- France
{Ileana.Ober, aboudib}@irit.fr

Abstract. We propose an approach to achieve interoperability in a family of domain specific language based on the use of their ASM semantics and of the category theory. The approach is based on the construction of a unifying language of the family, by using categorical colimits. Since the unifying language is obtained by construction, translators to this one are obtained easily. These are the premises for using ASM tools for symbolically executing systems made of components specified in domain specific languages of a same family.
Keywords: Family of domain specific languages, category theory, ASM, interoperability, Specware, CoreASM

1 Introduction

In previous work [1] we introduced an approach to achieve interoperability within a family of domain specific languages, by means of automatic unification of the considered languages. For this, we consider the category of the algebraic semantics of domain specific languages expressed in terms of algebraic specifications. Classical results in category theory, allow us to obtain *by construction* the formal semantics of a *unification language* as well as *translators* from the source DSLs to this *unification language*. Moreover, *properties* established in the context of DSLs and expressed as invariants, pre-conditions or post-conditions in the algebraic structure can be *transferred* to the unification language.

Our work starts from a case study that we developed with colleagues from the French Space Agency (CNES). This case study revealed their need to deal with a set of related – yet different – domain specific languages in remotely controlled satellites. Here, one main challenge is to handle the heterogeneity. The work in [1] addresses this problem, but the resulting framework lacks symbolic execution features. In order to get to a framework with symbolic execution, we apply a similar approach on the ASM specifications of programs specified in DSLs of a same family of languages. This leads to a framework in which a set of components specified in a family of DSLs could be translated to language unifying the family, using automatic translators.

This paper is composed of a brief overview of our general approach, the presentation of our current work around ASM specification of DSLs.

2 Unification of a family of DSL languages

Our thesis is that the interoperability within a family of DSLs can be rigorously tackled, by using a categorical approach. Our approach starts from the formal semantics of the various DSLs of a family. We consider category of algebraic specifications of the DSLs in a family. As recalled in [10], the algebraic specifications form a family. Using results from Category Theory 5, we can obtain – by using the *push-out* Categorical operator – in a quasi-automatic manner, the formal definition of a unifying language that combines the constructs from the various languages. The good thing about obtaining the unifying language in this manner is that it offers a "smart" union of the concepts existing in the several languages, i.e. it avoids duplication of concepts and it identifies correctly similar or related concepts.

In order to meet the hypothesis of applying colimits, we need to provide a boot language and a set of initialization morphisms. This corresponds to the fact that we have to express somewhere the correspondence between related constructs from various languages, in order to avoid the multiplication of similar concepts. The boot object and the initialization morphism play precisely this role and are the price to pay for obtaining, by construction, the unification language and the translation morphisms.

In a verification and validation setting, we can take advantage of the formal foundation of the unifying language definition. The pushout that leads to the unification language also preserves properties established in the context of the original languages 7. Therefore, it is possible to prove for a DSL *D* a property *P* given in an equational form, *P* is a thesis of *D*'s axioms. Thanks to our categorical approach, *P* is also verified in all other DSLs *E* connected from *D* through morphisms: the image of a thesis of *D* is a thesis of *E by construction*.

On the practical side, as illustrated in [1] the framework overviewed above was experimented by using Specware [4, 10]. This tool gives us a framework to reason on categories. We used it to obtain the unifying language of a family by pushouts on the category of algebraic specifications of DSLs in a family. This software supports proofs obligations written in higher-order logic. We use this to establish properties in the context of individual language specification. Moreover, in the context of Specware, proofs can be performed with the aid bridges with of theorem provers, such as Isabelle [8].

3 Adapting the framework to the use of ASM specifications

One natural extension of the approach summarized above is to get to a framework where the unified specifications could be used together in a framework supporting symbolic execution. This naturally led us to the abstract state machines [2].

In order to bring ASMs into the picture of the framework described before, we need to advance both on the theoretical and on the practical side.

On the theoretical side, the set of ASM specifications of the languages in the family does not lead to the category of algebraic specifications. Therefore, we have to identify a good category on which to reason on. Existing results on *especs* [9] show that it is possible to use categorical results and existing tools based on categories, with specifications using state machines.

We started on the practical side, by doing small experiments consisting in translating by hand ASM toy specifications in CoreASM [5] corresponding to specifications in source DSLs into the algebraic form accepted by Specware, in order to calculate pushouts for unifying them. Although most of the work is done by hand, we managed to identify some patterns, which we intend to automate. We are therefore confident, that a unification approach between ASM specifications resulting from various DSLs in a family is possible, and could be automatised with the aid of existing tools, such as Specware [4, 10], ASF+SDF [2].

4 Conclusions

This ongoing work continues the approach overviewed in [1], in an attempt to achieve executability in the context of specifications made in DSLs of a same family. Our aim is on one hand to get to a theoretical framework in which the ASM specifications can be unified by pushout, in order to take advantage of classical features of the ASM tools in order to symbolically execute specifications originally made in DSLs of a same family. The preliminary experiments validate this approach.

References

1. A. Abou Dib, L. Féraud, I. Ober, C. Percebois *Towards a rigorous framework for dealing with domain specific language families*, ICTTA, IEEE Computer Society, 2008
2. E. Börger, R. F. Stärk: *Abstract State Machines. A Method for High-Level System Design and Analysis* Springer 2003
3. M.G.J. van den Brand, A. van Deursen, J. Heering, H.A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P.A. Olivier, J. Scheerder, J.J. Vinj, E. Visser, J. Visser *The ASF+SDF Meta-Environment: a Component-Based Language Development Environment* Compiler Construction 2001, pp. 365-370, Springer Verlag, (2001).
4. Kestrel. Specware documentation. http://www.specware.org/doc.html
5. R. Farahbod, V. Gervasi, U. Glässer: *CoreASM: An Extensible ASM Execution Engine.* Fundamenta Informaticae 77(1-2): 71-103 (2007)
6. J.L. Fiadero. *Categories for Software Engineering,* Springer, (2005).
7. J. A. Goguen, R. M. Burstall. *Introducing Institutions: Abstract model theory for specification and programming*. Research Report ECS-LFCS-90-106, Univ. of Edinburgh
8. T. Nipkow, L. Paulson, M. Wenzel. Isabelle/HOL. A proof assistant for Higher Order Logic.Springer LNCS 2283, (2002)
9. D. Pavlovic, D. R. Smith: *Composition and Refinement of Behavioral Specifications*. ASE 2001, IEEE Computer Society: 157-165
10. D. Smith *Composition by Colimit and Formal Software Development* Algebra, Meaning, and Computation: A Festschrift in Honor of Prof. Joseph Goguen, LNCS 4060, 2006

UML-B: A plug-in for the Event-B tool set[1]

Colin Snook and Michael Butler

University of Southampton,
United Kingdom
{cfs,mjb}@ecs.soton.ac.uk

Abstract. UML-B provides a graphical front end for Event-B. It adds support for class-oriented and state machine modelling. UML-B is similar to UML but has its own meta-model. UML-B provides tool support, including drawing tools and a translator to generate Event-B models. The tools are closely integrated with the Event-B tools. When a drawing is saved the translator automatically generates the corresponding Event-B model. The Event-B verification tools (syntax checker and prover) then run automatically providing an immediate display of problems which are indicated on the relevant UML-B diagram. We introduce the UML-B notation, tool support and integration with Event-B.

UML-B is a graphical formal modelling notation based on UML [1]. It relies on Event-B [2] for its underlying semantics and is closely integrated with the Event-B verification tools [3]. UML-B and Event-B are implemented within the Eclipse [4] environment. This paper gives a brief introduction to UML-B. A more detailed description is provided in [5].

The UML-B modelling environment consists of a UML-B project containing a UML-B model. A builder is associated with the project and runs whenever the model is saved. Four interlinked diagram types (package, context, class and state machine) are provided. The top-level package diagram is opened with an empty canvas by the model creation wizard. This canvas represents the UML-B project. Package Diagrams are used to describe the relationships between top level components (machines and contexts) of a UML-B project. As in UML, package diagrams provide a structuring of the model, but also cater for the concept of refinement. The diagram shows the *refines* relationships between Machines, the *extends* relationships between Contexts and the *sees* relationships from machines to contexts. Other diagram types are linked and opened via model elements as they are drawn on the various canvases.

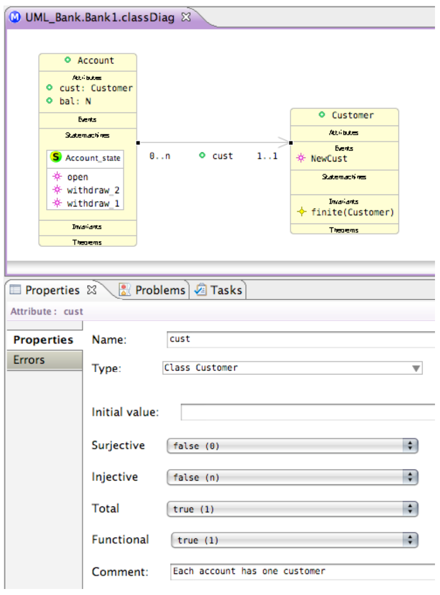
UML-B mirrors the Event-B approach where static data (sets and constants) are modelled in a separate package called a *'context'*. The Context diagram defines the static (constant) part of a model. The context diagram is similar to a class diagram but has only constant data represented by *ClassTypes*, *Attributes* and *Associations*. *Axioms* (given properties about the constants) and *Theorems* (assertions requiring proof) may be attached to the ClassTypes. ClassTypes define 'carrier' sets or constant

[1] This work was carried out under the EU projects, Rodin [IST-511599] and ICT project Deploy [IP-214158].

subsets of other ClassTypes. ClassTypes may own immutable attributes and associations which represent constant functions with the ClassType as domain.

The behavioural parts (variables and events) are modelled in a Class diagram which is used to describe the *'machine'*. Classes represent subsets (variable or fixed) of the ClassTypes that were introduced in the context. The class' associations and attributes are similar to those in the context but represent variables instead of constants.

The correspondence between an association's multiplicity constraints and the constraints on the resulting Event-B relationship is clear from the drawing tool. An example Class diagram with an association selected and shown in the properties view is given in Fig. 1.



Attribute : cust

Name: cust

Type: Class Customer

Initial value:

Surjective: false (0)

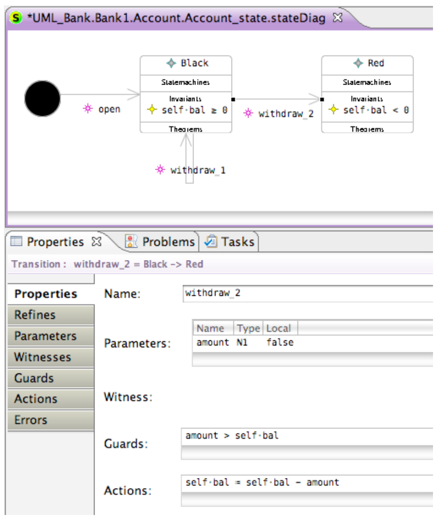
Injective: false (n)

Total: true (1)

Functional: true (1)

Comment: Each account has one customer

Fig. 1. Class diagram



Transition : withdraw_2 = Black -> Red

Name: withdraw_2

Parameters:

Name	Type	Local
amount	N1	false

Witness:

Guards: amount > self·bal

Actions: self·bal ≔ self·bal − amount

Fig. 2. Statemachine diagram

Classes may own *events* that modify the variables. Event *parameters* can be added to an event providing local variables to be used in the transition's guards and actions. These parameters can be used to model inputs and outputs. Class events implicitly utilise a parameter to non-deterministically select the affected instance of the class. This instance is referred to via the reserved word self when referencing the attributes of the class.

State machines may be used to model behaviour. Transitions represent events with implicit behaviour associated with the change of state. The event can only occur when the instance is in the source state and, when it fires, the instance changes to the target state. Hence statemachines model a class variable similar to an attribute. Additional guards and actions can be attached to the transition in the property view. An example statemachine is shown in Fig 2. A transition is selected and its properties, including additional guards and actions, are shown in the properties view.

Conclusion

UML-B is a fully integrated graphical front end for Event-B. UML-B retains sufficient commonality with UML for the main goals of approachability to be attained by industrial users. Since UML-B automates the production of many lines of textual B, models are quicker to produce and hence exploration of a problem domain is more attractive. This assists novices in finding useful abstractions for their models. We have found that the efficiency of UML-B and its ability to divide and contextualise mathematical expressions, assists novices who would otherwise be deterred from writing formal specifications. Furthermore, UML-B is gaining acceptance as a useful visual aid for more experienced formal methods users.

UML-B has been used to model a failure management system (FMS) [6]. The FMS is a wrapper layer that detects and filters out transient failures in sensors and transducers. In the FMS case study we used UML-B to specify the generic problem domain in an entity-relationship style that could be instantiated with specification objects to 'configure' the specification for a particular application. UML-B was found to be very suitable for this kind of problem.

Several groups have investigated UML based graphical renderings of B [7, 8] as well as our own previous work [9]. However, our work is unique in providing a link to Event-B and the first to provide a tool highly integrated with strong formal proof tools. Our work also differs by defining its own language which has avoided many of the problems highlighted in previous work.

References

[1] G. Booch, I. Jacobson, and J. Rumbaugh, *The unified modeling language - a reference manual* (Addison-Wesley, 1998).

[2] C. Métayer, J.R. Abrial and L. Voisin, Event-B Language, *RODIN Deliverable D7* [Rodin], 2005.

[3] J.R. Abrial, S. Hallerstede, F. Mehta, C. Métayer and L. Voisin, Specification of Basic Tools and Platform, *RODIN Deliverable D10* [Rodin 2005].

[4] J. D'Anjou, S. Fairbrother, D. Kehn, J. Kellerman, P. McCarthy, *The Java developer's guide to Eclipse, 2nd Edition* (Addison-Wesley, 2004).

[5] C. Snook and M. Butler, UML-B and Event-B: An integration of Languages and Tools, *Proceedings of the IASTED International Conference Software Engineering. SE2008,* ISBN:978-0-88986-715-4.

[6] C.Snook, M. Poppleton and I. Johnson, Rigorous engineering of product-line requirements: a case study in failure management, *Information and Software Technology*, In press (available on-line 26 Oct 2007).

[7] K. Lano, D. Clark, and K. Androutsopoulos, UML to B: formal verification of object-oriented models, *Proc. Integrated Formal Methods, 4th International Conference, IFM 2004*, LNCS Vol. 2999 Springer, 187-206.

[8] H. Ledang and J. Souquières, Integrating UML and B specification techniques, *Proc. Informatik2001 Workshop on Integrating Diagrammatic and Formal SpecificationTechniques,* 2001.

[9] C. Snook and M. Butler, Formal modeling and design aided by UML, *ACM Transactions on Software Engineering and Methodology (TOSEM), Volume 15*(1), 2006, 92 – 122

Tool Support for the *Circus* Refinement Calculus

A. C. Gurgel, C. G. de Castro and M. V. M. Oliveira

Departamento de Informática e Matemática Aplicada, UFRN, Brazil

Circus [1] specifications combine both data and behavioural aspects of concurrent systems using a combination of CSP [3], Z [9], and Dijkstra's command language. Its associated refinement theory and calculus [5] distinguishes itself from other such combinations. Using the *Circus* refinement calculus, we can correctly construct programs in a stepwise fashion [4]. Each step is justified by the application of a refinement law, possibly with the discharge of proof obligations (hereafter called POs). Hence, using *Circus* we are able to calculate concrete (usually distributed) specifications from abstract (usually centralised) specifications. The manual application of the refinement calculus, however, is an error-prone and hard task.

We present CRefine[1], a tool that supports the use of the *Circus* refinement calculus[2]. Its interface is similar to Refine's [8], a tool that supports Morgan's refinement calculus [4]; it is based on an early prototype that was presented in [10]. We have, however, considerably changed and extended CRefine's prototype. First, we updated the *Circus* parser used which fixes a couple of bugs of its earlier version. We have also added facilities to manage developments: undoing and redoing refinement steps, saving and opening developments is now available. Furthermore, some GUI facilities like pretty-printing, filtering applicable laws according to the selected program, classification of laws, adding comments to the development, and printing the development were also included. Finally, the discharge of some proof obligations is now automatically done by CRefine.

CRefine provides support to apply the refinement laws and to manage the overall development. Its **interface** is composed by a menu and three main frames: refinement, proof obligations, and code. The refinement frame shows all the steps of the refinement process. This includes law applications and retrieving the current status of an action or process (collection). The proof obligations frame lists the POs that were generated by the law applications, indicates their current state (i.e. checked valid or invalid, or unchecked), and associates each one of them to the law application that originated it in the refinement frame. Currently, some proof obligations are automatically checked valid or invalid. In our experience, these amount to over 60% of the proof obligations. The remaining proof obligations need to be verified by the user. Finally, the code frame exhibits the overall *Circus* specification that has been calculate so far.

CRefine provides two display formats for formulas: LaTeX and Unicode (pretty-printing). We intend to use this tool in teaching the *Circus* refinement calculus to under-graduates. Unfortunately, most of them are not familiar with LaTeX; in order to make CRefine accessible to them, we have also provided a pretty-printing.

[1] Available at `http://www.cs.york.ac.uk/circus`

[2] This work is financially supported by CNPq: grant 551210/2005-2

This pretty-printing is also a success among researchers, since it unconditionally makes the presentation of the development more user-friendly.

The starting point of a **development** in CRefine is a LaTeX file that contains the abstract specification of the system to be refined. Starting from this specification, the application of refinement laws is as follows: first, we select the part of the *Circus* program that we want to refine by clicking on its lines (multiple lines can be selected by clicking on the first and last line of the term); then, we select the law we want to apply; finally, after the input of any arguments that may be required by the law, the application is automatically done. This updates the refinement frame, the proof obligations frame, and the code frame.

Law applications can be done in two ways. First, a right-click on the selected term shows a pop-up menu that lists only those laws that can be applied to the selected term. In this case, their effective application is done by selecting them in this list. Second, we can select the law from a list in the main menu that contains all the refinement laws. If the law can be applied to the term, the apply button is enabled and we can apply the law by clicking on it. When needed, an argument window is shown to the user before the application of the law. In this window, users may either type the argument in a LaTeX format, or use a symbol keyboard. The user may see the details of a refinement law by selecting it in the list of the main menu and then right-clicking on its name.

Using CRefine's **development management** users may (when applicable) undo and redo development steps. That means different development paths during a development may be tried, possibly in a search for a more efficient implementation. Developments may also be saved in order to be continued latter; CRefine's developments are saved in XML format. Users may document the development by adding, editing, and viewing comments to each term or law application in the refinement window. Finally, CRefine automatically generates a LaTeX file that documents the main elements of the development: original specification, refinement, POs, comments and the concrete specification. The user can choose which elements should be included in the final document.

CRefine's **architecture** is strongly based on the architecture proposed for *Circus* tools in [2]. It extends an ongoing effort of the Community Z Tools (CZT), which provides a set of tools for the Z specification language. In recent years, many *Circus* collaborators have made extension on the CZT project to provide tools that support *Circus* like a parser, a type-checker, a refinement model-checker, a theorem-proving module, and pretty-printers.

The cost of developments may still be reduced. Frequently used strategies of refinement are reflected in sequences of laws that are applied over and over again. Identifying these strategies, documenting them as tactics, and using them in program developments as single transformation rules brings a profit in effort. In [6], we present a refinement-tactic language called ArcAngel*C*, which can be used to formalise tactics of refinement just like in [7]. Allowing users to define and use tactics as simple refinement laws within CRefine is our next step.

The vast majority of the refinement laws from [5], which have been used in a reasonable number of case studies, are included in CRefine. This give us confidence that the current set of laws is appropriate for useful applications. We are aware, however, that it is not complete [5]. We intend to provide a parser of refinement laws in the style of CZT. Using this parser, the laws could be dynamically loaded; no recompilation would be needed.

Another interesting piece of future work is the automatic discharge of the remaining POs, which can be predicates or action/processes transformations. For this, we need to integrate CRefine with a theorem-prover to check predicate POs and to allow multiple developments within CRefine to check POs that are action/processes transformations. For instance, users will be able to prove that A_1 is refined by A_2 by deriving A_2 from A_1 in a new development.

Finally, the infra-structure provided by tactics of refinement and multiple developments can be used to allow users to make sub-developments within larger developments. This would considerably modularise future developments.

CRefine can be a useful tool in the development of state-rich reactive systems. Our initial intention was to develop an educational tool and use it in teaching formal methods. However, during the implementation and tests, we noticed that it may as well be useful in the development of industrial-scale systems. Empirical verifications in a near future will verify this statement. For instance, we are currently developing case studies that are related to the oil industry.

References

1. A. L. C. Cavalcanti, A. C. A. Sampaio, and J. C. P. Woodcock. A refinement strategy for *Circus*. *Formal Aspects of Computing*, **15**(2–3):146–181, 2003.
2. L. J. S. Freitas, J. C. P. Woodcock, and A. L. C. Cavalcanti. An Architecture for *Circus* Tools. In A. C. V. Melo and A. Moreira, editors, *Proceedings of the Brazilian Symposium on Formal Methods*, pages 6 – 21, 2007.
3. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
4. C. Morgan. *Programming from Specifications*. Prentice-Hall, 1994.
5. M. V. M. Oliveira. *Formal Derivation of State-Rich Reactive Programs using Circus*. PhD thesis, Department of Computer Science, University of York, 2005. YCST-2006/02.
6. M. V. M. Oliveira. ArcAngel*C*. Technical report, Departamento de Informática e Matemática Aplicada - Universidade Federal do Rio Grande do Norte, Natal, Brazil, February 2007.
7. M. V. M. Oliveira, A. L. C. Cavalcanti, and J. C. P. Woodcock. ArcAngel: a Tactic Language for Refinement. *Formal Aspects of Computing*, **15**(1):28–47, 2003.
8. M. V. M. Oliveira, M. Xavier, and A. L. C. Cavalcanti. Refine and Gabriel: Support for Refinement and Tactics. In Jorge R. Cuellar and Zhiming Liu, editors, *2nd IEEE International Conference on Software Engineering and Formal Methods*, pages 310–319. IEEE Computer Society Press, Sep 2004.
9. J. C. P. Woodcock and J. Davies. *Using Z—Specification, Refinement, and Proof*. Prentice-Hall, 1996.
10. M. A. Xavier, A. L. C. Cavalcanti, and A. C. A. Sampaio. Type Checking *Circus* Specifications. In A. M. Moreira and L. Ribeiro, editors, *SBMF 2006: Brazilian Symposium on Formal Methods*, pages 105 – 120, 2006.

Formal Verification of ASM Models using TLA$^+$

Hocine El-Habib Daho[*] and Djilali Benhamamouch

Department of Informatics
University of Oran, Algeria
[*]dahoh@yahoo.com

Abstract. The notion of Abstract State Machines(ASMs) handles a practical new approach for modeling and analysing various kinds of discrete dynamic systems. In the context of the verification problem of ASM models, formal verification techniques based on variants of restricted first-order temporal logic have been used to verify correctness of restricted forms of ASM specifications. In this spirit, the current work shows how the state-based logic of TLA$^+$can be employed to formally reason about dynamic systems formalised in terms of ASMs.

1 Introduction

Abstract State Machines(ASMs), previously called Evolving Algebras and introduced by Y.Gurevich in[9], constitue the formal foundation of a practical methodology in modeling and analysing various kinds of complex dynamic systems.

The ASMs formal method has been applied in different areas, such as software and hardware systems, programming languages, communication protocols and distributed algorithms[3]. It provides a flexible formalism to specify the operational semantics of a system at natural abstraction level in a direct and intuitive way[3].

The ASM approach belong to the family of state-based methods, which model a system as a transition system. An ASM model describes the state space of a system by means of universes(i.e. basic sets) with functions and relations interpreted on them, and the state transitions by means of transition rules. In applications, Abstract State Machines are considered a suitable specification formalism for giving semantics of a system in terms of its set of possible executions(i.e state sequences).

Besides the standard mathematical techniques underlying the ASM approach that naturally support verification of ASM model properties, there has been work on formal proof systems for ASMs, using various formal verification tools[7, 14, 16]. Other work about verification of ASM models include[13, 15] who investigated the problem of verifying classes of restricted ASM programs(called Nullary ASMs and guarded ASMs) automatically. [6] show how to translate ASM specifications into the first-order temporal logic(FOTL), especially into the monodic fragment of FOTL. They have defined restrictions on ASM specifications which ensure that the temporal translation is in a decidable fragment of FOTL. Work in[8] introduces the formal language for ASMs (called FLEA), a system for formal reasoning about ASM specifications. They have adopted a modal logic view. [1] presents work on the specification and verification of real-time systems within a logical framework where ASM formalism(e.g. a Block ASM) is used to specify timed algorithms. They have used a type of first-order timed logic(FOTL) to formally specify and reason about behavior of real-time ASM specifications by means of FOTL-formulas.

2 Reasoning about ASMs within the TLA$^+$-Logical Framework

In our ongoing research work[4, 5], we propose to adopt Lamport's Temporal Logic of Actions(TLA)[10] as an appropriate alternative to the logic-based approaches [1, 6, 8, 13, 15], to formally reason about ASM specifications of dynamic systems without imposing restrictions on the specification formalisms. TLA is a state-based logic which provides the means for describing transition systems(i.e states, state transitions and thereby the resulting state sequences) and formulating their properties in a single logical formalism, equipped with a relatively complete set of proof rules for reasoning about safety and liveness properties that can be required for systems.

The operational behavior(semantics) of an ASM specification is directly defined by TLA-logical formulas and the TLA-proof techniques can be applied to formally prove the correctness of ASM specifications. Using this framework, both ASM specifications and required properties are represented by formulas in the same logic.

In particular, we provide some basic rules to translate ASM models into TLA$^+$specifications. TLA$^+$is a formal specification language based on Zermelo-Fränkel set theory, first-order logic and the linear-time temporal logic TLA[11]. In addition to the operators of TLA, it contains operators for defining and manipulating data structures and syntactic structures for handling large specifications. The TLA$^+$ framework offers a potential mathematical framework into which ASM model elements are directly translated to their most natural equivalents in TLA$^+$.

The applicability of the proposed TLA$^+$approach is illustrated by the formal correctness proofs of both Lamport's bakery algorithm and a token ring algorithm both formalised in terms of ASMs[2, 12].

3 Conclusion and Further Work

The aim of our current work is to provide formal reasoning techniques for ASMs by using the TLA$^+$-logical framework. In this work we show how the proposed TLA$^+$approach can be used to formally reasoning about the correctness of ASM specifications. We started the work by hand-translating examples of ASM specifications describing distributed algorithms into TLA$^+$specifications. Invariance and liveness properties were verified for these specifications using formal hand proofs written in the TLA$^+$logic. Based on our hands-on experience, we came up with a scheme for translating ASM specifications to TLA$^+$specifications. Futur work will concentrate on the development of a model translator, namely ASM2TLA$^+$ translator, to perform the translation of an ASM model into a TLA$^+$model which can be verified automatically using the TLA$^+$model checker called TLC[11].

References

1. Beauquier,D.,Slissenko,A.: A first-order logic for specification of timed algorithms : Basics properties and a decidable class. Annals of Pure and Applied Logic, 113(1-3):13-52,2002.
2. Börger,E.,Gurevich,Y.,Rosenzwerg,D.: The bakery algorithm : Yet another specification and verification. In E.Bröger(ed.), Specification and Validation Methods, pages 231-243. Oxford University Press. 1995.
3. Börger,E., Stärk,R.: Abstract State Machines : A Method for High-Level System Design and Analysis. Springer 2003.
4. El-Habib Daho,H.,Benhamamouch,D.: Verifying the correctness of ASM Programs using TLA$^+$. Technical Report, Department of Informatics, Oran's University, January 2008.(In French)
5. El-Habib Daho,H.,Benhamamouch,D.: Specification and verification of ASM models with TLA$^+$ and TLC. Technical Report, Department of Informatics, University of Oran, March 2008.(In French)
6. Fisher,M.,Lisitsa,A.: Monodic ASMs and temporal verification. In Proceedings of Abstract State Machines Workshop ASM'2004, May 2004.
7. Gargantini,A.,Riccoben,E.: Encoding Abstract State Machines in PVS. In Abstract State Machines : Theory and Applications, Vol.1912 of LNCS.Springer-Verlag, 2000.
8. Groenboom,R.,Lavalette,G.R.: A Formalisation of Evolving Algebras. In Proceedings Accolade 95, pages 17-28, 1995.
9. Gurevich,Y.: Evolving Algebras : An attempt to discover semantics. Bulletin of the EATCS, (43):264-284, February 1991.
10. Lamport,L.: The Temporal Logic of Actions. ACM Transactions on Programming Languages and systems, 16(3):872-923, May 1994.
11. Lamport,L.: Specifying Systems : The TLA$^+$Language and Tools for Hardware and Software Engineers. Addison-Wesley, Boston 2003.
12. Matousek,P.: Evolving Algebras and Formalisation of Dynamic Systems. Technical Report-VSB-TU, Department of Informatics, VSB Technical University of Ostrav, Czech Republic, 1999.
13. Nowack,A.: Deciding the Verification Problem for Abstract State Machines. In Proceedings of ASM 2003, LNCS, 2589:341-371, 2003.
14. Schellhorn,G.,Ahrendt,W.: Reasoning about Abstract State Machines : The WAM case study. Journal of Universal Computer science,3(4):377-413, 1997.
15. Spielmann,M.: Automatic verification of Abstract State Machines. In Proceedings of 11th international conference on computer-Aided Verification(CAV'99), Vol.1633 of LNCS, pages 431-442. Springer-Verlag, 1999.
16. Winter,K.: Model Checking for Abstract State Machines. Journal of Universal Computer Science, 3(5):689-701, 1997.

BART: A tool for automatic refinement[⋆]

Antoine Requet

antoine.requet@clearsy.com
ClearSy
Parc de la Duranne
320, avenue Archimde
Les Pliades III - Bt A
13857 AIX EN PROVENCE CEDEX 3 - FRANCE

Abstract. This paper provides an introduction to BART (B Automatic Refinement Tool), a tool for automatizing the refinement of B machines. The BART tool is currently in development, and will be integrated within the next major version of the Atelier B tool.

1 Introduction

Refining a B specification into an implementation can be a complex and time consuming process. This process can usually be separated in two distinct parts:

- the specification part, where the refinement is used to introduce new properties and specification details, and
- the implementation, where refinement is used to convert a detailed B specification into a B0 implementation.

The first part requires human interaction, since it corresponds to writing the specification. However, the implementation part is more mechanical, and usually corresponds to apply known refinement schemes.

The BART tool aims to provide helps for this second part of the B development, by automatically refining machines or refinements to B0 implementations. It uses the same approach as the tool described in [LB99] that has been developed by Siemens.

2 Automatic Refinement

The purpose of the automatic refinement is to generate B implementations from a B specification or refinement. To provide a meaningful output, the tool requires the refined specification to be detailed enough: complex refinements designed to prove properties, or to add specification details still have to be done manually.

To refine a specification, the tool performs the two following refinements:

⋆ This work has been funded by french "Agence Nationale de la Recherche" ANR-06-SETI-015.

– Refinement of data: abstract variables are refined to concrete variables.
– Refinement of algorithms: the substitutions used within the abstract machines are refined into equivalent B0 substitutions.

In its simplest form, the refinement of a B component produces a corresponding B implementation. However, in most case, the result will be a set of machines and implementations, as the refinement rules will split complex refinements into several machines.

The proof of the refined machines remains to be done using the Atelier B tool. This simplifies the customisation of the refinement rules, as an incorrect refinement will be detected when proving the generated implementations.

2.1 Principles

The tool uses a set of refinement rules, that are checked against the refined machine. Additionally, annotations can also be added to the B model in order to modify the behavior of the tool. For instance it is possible to specify that a variable should be refined by a specific refinement rule.

The syntax of BART's refinement rules is a superset of the syntax used by the tool described in [LB99].

Three categories of rules are used by BART:

– variable rules: those rules specify how an abstract variable should be implemented.
– operation rules: those rules are used to refine and implement substitutions.
– initialisation rules: those rules are similar to the operation rules, but are specialised for the initialisation clause.

All those rules work by pattern-matching, and by looking for required hypothesis. They mainly contain three elements:

– a pattern indicating which kind of element can be refined by the rule. For operation and initialisation rules, this pattern corresponds to a substitution. For variables, the pattern is matched to the variable name.
– a set of constraints that must be met for the rule to be applied. Those constraints are checked against the invariants and properties visible by the refined machine.
– a pattern describing the refined element. For operation and initialisation rules, this pattern describes the refined substitution. For variable rules, this pattern contains the introduced variable as well as the gluing invariant.

Those rules are then applied to the refined machine until no rule can be applied, or the machine is refined into a valid implementation.

3 Conclusion

The benefits of using an automatic refinement tool can be listed as follows:

- The most obvious advantage is that it automatises repetitive tasks.
- Additionally, the proof of the generated machines is usually simpler, since all the generated machines share the same model.
- finally, it is a way of reusing refinement patterns, and capitalizing on refinement experience.

The BART tool is currently in development and will be integrated in the next major version of Atelier B.

References

[LB99] L. Burdy, J-M. Meynadier, Automatic Refinement, FM'99 workshop – Applying B in an industrial context : Tools, Lessons and Techniques, Toulouse, 1999

A roadmap for the Rodin toolset[⋆] Version 1.0: 12 June 2008

Jean-Raymond Abrial[1], Michael Butler[2], Stefan Hallerstede[2], and Laurent Voisin[3]

[1] ETH Zurich, Switzerland, `jabrial@inf.ethz.ch`
[2] University of Southampton, United Kingdom, `{mjb,sth}@ecs.soton.ac.uk`
[3] Systerel, France, `laurent.voisin@systerel.fr`

1 Event-B and the Rodin Platform

Event-B is a formal method for system-level modelling and analysis [1]. Key features of Event-B are the use of set theory as a modelling notation, the use of refinement to represent systems at different abstraction levels and the use of mathematical proof to verify consistency between refinement levels.

The Rodin Platform[4] [2] is an Eclipse-based [3] toolset for Event-B that provides effective support for refinement and mathematical proof. Keep aspects of the are

– support for abstract modelling in Event-B
– support for refinement proof
– extensibility
– open source

To support modelliing and refinement proofs Rodin contains a modelling database surrounded by various plug-ins: a static checker, a proof obligation generator, automated and interactive provers. The extensibility of the platform has allowed for the integration of various plug-ins such as a model-checker (ProB), animators, a UML-B transformer and a LaTeX generator. The database approach provides great flexibility, allowing the tool to be extended and adapted easily. It also facilitates incremental development and analysis of models. The platform is open source, contributes to the Eclipse framework and uses the Eclipse extension mechanisms to enable the integration of plug-ins.

2 Roadmap

In its present form, Rodin provides a powerful and effective toolset for Event-B development and it has been validated by means of numerous medium-sized case studies. Naturally further improvements and extensions are required in order to improve the productivity of users further and in order to scale the application of the toolset to large industrial-scale developments. We outline the main extensions to Rodin that we have planned for a four year time frame. The outline descriptions of these planned extensions are grouped into sections 2.1 to 2.5 as follows.

[⋆] The continued development of the Rodin toolset is funded by the EU research project ICT 214158 DEPLOY (Industrial deployment of system engineering methods providing high dependability and productivity) www.deploy-project.eu. The toolset was originally developed as part of the project IST 511599 RODIN (Rigorous Open Development Environment for Complex Systems).
[4] Available from www.event-b.org

2.1 Model construction

Rodin provides a structured editor for constructing and modifying models stored in the database. As mentioned above, this facilitates easy extension as well as incremental development and analysis of models. Rodin needs further improvement to make it easier to perform standard editing tasks such as text search, copy/paste and undo/redo. Rodin will be extended to provide refactoring facilities, such as identifier renaming, that can be applied not just to models but to proof obligations, proofs and other forms of elements. Better support for browsing refinement links between models will be provided, for example, allowing the refinements and abstractions of events to be followed down or up a refinement chain.

2.2 Scaling

Event extension: In many Event-B developments it is common to perform superposition refinement where existing model features are maintained and additional features are added (e.g., additional variables, invariants, events and additional guards and actions for existing events). Currently events can be inherited as a whole but not extended. Rodin will support event extension (or superposition) where only the additional features are defined in a refined event and the existing features are inherited.

Composition and decomposition: Composition and decomposition of models is essential for scalability. There are plans to support two styles of composition for Event-B in Rodin:

Style A Sub-models interact via shared variables
Style B Sub-models interact via synchronisation over events

Rodin will be extended to provide support for composing models as well as decomposing models according to these styles. The proof obligation generator will be extended to enable independent refinement of sub-models.

Team-based development: Support for composition and decomposition will go some way towards enabling team-based development. But there will still be situations where a team needs to access a common set of models. Rodin will be extended to support concurrent modification of developments by providing viewing of change conflicts and automated merge of changes. It will provide support for version control. Support to analyse the impact of multiple user modifications on proof will be investigated.

2.3 Extending the proof obligations and theory

Proof obligations: Event-B models will be extended to include external variables. The proof obligation for such variables is that they must be preserved via a functional gluing invariant between abstract and concrete external variables. Other forms of proof obligations will also be added to support different paradigms (concurrent, distributed, sequential systems). These include proof obligations for preservation of event enabledness and richer variant structures(such as pointwise ordering and lexicographic ordering) for convergence proof obligations.

Mathematical extensions: Rodin will be extended to support richer types such as record structures and user-defined data types including inductive data types. Appropriate automated and interactive proof support for richer types will be investigated and provded. Higher order provers should enable proof support for inductive datatypes. Users will be able to define operators of polymorphic type (but not use operator overloading) as well as parameterised predicate definitions. Support for disjointness constraints will be added.

2.4 Proof and model checking

Rodin provides an open architecture for proof in the form of a proof manager that can use a range of provers to discharge proofs and sub-proofs. The existing automated provers will be extended with more powerful decision procedures. The use of existing first order and higher order automated provers will be investigated. As mentioned already, higher order provers should enable proof support for inductive datatypes. The possibility of exploiting automated techniques such as SMT and SAT will be investigated. The facilities of the ProB model checker will be fully integrated into Rodin.

2.5 Animation

Prototype animation plug-ins already exist. The animation facilities will be extended to allow for greater automation of large animations to support regression testing of models. A clear API to the animation will be provided to allow for easy integration with graphical animation tools.

2.6 Process and productivity

Requirements Handling and Traceability: The interplay between informal requirements and formal modelling is crucial in system development and needs better tool support. Facilities for constructing structured requirements documents and for building links between informal and formal elements will be added to Rodin. These will support traceability between requirements and formal models. Support for recording validation of these links and for managing consistency under change to requirements and to formal models will be provided.

Document management: Currently, the B2Latex plug-in for Rodin generates a LaTeX version of an Event-B model. The structure of the document follows the structure of the model. For proper document generation tool support will be provided whereby users dictate the order in which parts of the model are presented. They should be able to write a document, structured according to their needs that includes parts of an Event-B project and that is automatically kept in synchrony with the models.

Automated model generation: Automatic generation of refinements will be investigated and appropriate tool support provided. More general modelling and refinement patterns, enabling greater reuse of modelling and refinement idioms, will be investigated and tool support provided. Code generation from models will be investigated. An indirect route for achieving code generation will be to generated classical B and use the existing code generators for classical B.

3 Tool development procedures

We are promoting a rigorous approach to the development of Rodin. Key features of Rodin, e.g., the static checker and the proof obligation generator, were specified formally before being implemented. Test procedures are developed in tandem with implementing. We are setting up rigorous specification and code review procedures. The development of new features should also follow this approach.

Many of the extensions listed above will first be implement as separate Eclipse plug-ins. When their general value and quality is assured, they will be incorporated into the platform release.

The management of platform release versions will be coordinated amongst the platform and plug-in developers. Facilities for importing existing developments into newer versions of Rodin will be provided. Support documentation and tutorial material for tool users and plug-in developers will continue to be improved and updated.

4 Concluding

Many of the Rodin extensions outlined above will be implemented as part of the DEPLOY project. However, we welcome support from other researchers and tool developers in elaborating and realising the roadmap. Furthermore, we anticipate that researchers will investigate and implement Rodin extensions that are not identified in our roadmap.

References

1. Jean-Raymond Abrial. Modelling in Event-B: System and software engineering. To be published by Cambridge University Press, 2008.
2. Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, and Laurent Voisin. An open extensible tool environment for Event-B. In Z. Liu and J. He, editors, *ICFEM 2006*, volume 4260, pages 588–605. Springer, 2006.
3. Erich Gamma and Kent Beck. *Contributing to Eclipse*. Addison Wesley, 2003.

Acknowledgements

We would like to thank all members of the RODIN and DEPLOY projects and others who are contributing to the toolset especially Jens Bendisposto, Dominique Cansell, Kriangsak Damchoom, Thai Son Hoang, Cliff Jones, Thierry Lecomte, Michael Leuschel, Farhad Mehta, Christophe Métayer, Colin Snook and Francois Terrier.

Model Checking Event-B by Encoding into Alloy[1]

Paulo J. Matos and João Marques-Silva

Electronics and Computer Science, University of Southampton
{pocm, jpms}@ecs.soton.ac.uk

Abstract. As systems become ever more complex, verification becomes more main stream. EVENT-B and ALLOY are two formal specification languages based on fairly different methodologies. While EVENT-B uses theorem provers to prove that invariants hold for a given specification, ALLOY uses a SAT-based model finder. In some settings, EVENT-B invariants may not be proved automatically, and so the often difficult step of interactive proof is required. One solution for this problem is to validate invariants with model checking. This work studies the encoding of EVENT-B machines and contexts to ALLOY in order to perform temporal model checking with ALLOY's SAT-based engine.

1 Introduction

Current day systems are ever more detailed and complex leading to the necessity of developing models that abstract unimportant implementation details while emphasizing their structure. These models are developed in order to be easily verified either by theorem provers or model checkers.

EVENT-B is a language based on B-METHOD [1], and supported by the open tool RODIN [2]. Part of the language is developed visually and there is a syntax for predicates and expressions[2]. The RODIN tool is shipped with theorem provers which allow the user to prove the model invariants either automatically or interactively.

ALLOY [3] is a structural modeling language based in first order logic. It is a textual language that consists of signatures, which introduce flat relations, functions, predicates and assertions that deal with the relations. By specifying predicates and assertions it is possible to perform model finding or model checking. The tool uses KodKod [4], a model finder, to convert the model within given bounds to SAT and return an instance of the model or a counter-example.

In previous work [5] B-METHOD was combined with ALLOY. since then ALLOY was considerably improved and extended by, not only, adding new constructs but also by making the verification much more efficient. The restrictions described in the work either do not exist anymore or they can be overcome.

Until recently it was only possible to perform temporal model checking in an EVENT-B model by using a two step process: converting the model to B-METHOD

[1] This work is partially supported by EPSRC grant EP/E012973/1, and by EU grants IST/033709 and ICT/217069.

[2] The RODIN toolkit was developed under research project IST 511599.

and then using ProB[6]. However, the solution was not straightforward as the conversion to B-METHOD was not always fully automated. More recently, a prototype ProB plugin[7] for the RODIN tool has been developed that provides an alternative solution for model checking EVENT-B models using the ProB model checker. Nevertheless, encoding EVENT-B to ALLOY allows building on top of the ALLOY model finding engine therefore benefiting from all of its optimizations. As mentioned elsewhere [7], encoding EVENT-B is not straightforward but this work shows it is possible.

2 Encoding

This section summarizes the process of encoding an EVENT-B model into ALLOY. Due to space constraints the presentation will be informal and far from complete, which means that there are EVENT-B operators for which an encoding has been developed but which are not described. A full report with an example based on processes and mutexes, including the actual encoding and additional comments is available elsewhere [8]. For a detailed description of the structure and rules of EVENT-B models the reader is referred to [2].

There are three aspects to the encoding: encoding of the model structures, expressions, and predicates (which are straightforward due given the existence of the logical operators in both languages).

The execution model needs to be emulated by the final ALLOY model. To this end all of the encoded models define a "State" signature which is ordered by using the ordering module and with as many fields are there are variables. The variable types are extracted from the set of EVENT-B invariants and encoded into signatures which become the type of the respective fields. A fact defines the initial state which is encoded from the "Initialisation" event. Each event is encoded in a predicate with two arguments: the current state and the next state and it evaluates to true whenever the next state reflects the triggering of an event from the current state. A final fact asserts that at every state one of the events needs to be triggered.

A carrier set is encoded as a signature with no fields and an enumerated set is encoded as signature, one per enumeration, with no fields that extend a base signature that represents the type of the enumerated set. One special enumerated set "Events" has one enumeration per event, plus an "Undef" enumeration for the initial state which is defined to be triggered by an "Undef" event. "Ev" is the type of a special field in the "State" signature. If "Ev" is x in state s', this means that it was the triggering of x that caused the transition from s to s'. Although this information is not necessary for the model checking itself, it makes the state trace of a failed invariant much more readable.

Expressions are the hardest part to encode. There is not only a myriad of complex expressions in EVENT-B but given that ALLOY uses only flat relations, some EVENT-B expressions that introduce relations with nested sets generate many (and potentially large) ALLOY expressions. Some expressions are straightforward like the domain, range, domain and range restriction and their subtraction counterparts, since they are either already defined in modules shipped as part of the current Alloy distribution or they are very easily defined as small functions. Operators like prj_1, prj_2 and id need to be defined as Alloy functions in order to be used. All arithmetic operators except power are defined but still, power can be defined explicitly during encoding-time depending on the defined bit width passed on to the Alloy engine for checking. For example, a^b could be defined as $a = 0 \Rightarrow 1$ else $b = 0 \Rightarrow 0$ else $b = 1 \Rightarrow a$ else $b = 2 \Rightarrow a.mult[a] \cdots$. Function expressions can be encoded as relations and then facts can be added to the model as to assure the semantics is preserved. So, to encode $(A \rightarrow B) \leftrightarrow C$, a signature with a relation from A to B would be defined followed by a fact asserting the relation to be a total function and then yet another signature is defined with a relation from the previously defined signature to C. Although function nesting requires a new signature, it seems this is the best general solution given that Alloy only works with flat relations.

3 Conclusion and Future Work

The focus of our work is to allow the users of the Event-B language to use the years of work and expertise in the development of the Alloy tools. This paper summarizes an encoding of Event-B into Alloy. The resulting Alloy model can serve to find counterexamples to false invariants and translate them back to Event-B. Future work entails the automatic generation of the encoding and its integration with the RODIN tool. The tool to be developed can then be extended to use other backends besides Alloy. Although this work focuses on model checking, there are cases where it is desirable, and often the preferred alternative, to do constraint based checking. We will investigate line of work as another possible extension to the tool.

References

1. Abrial, J.R.: The B-book: assigning programs to meanings. Cambridge University Press, New York, NY, USA (1996)
2. Consortium, R.: Rodin deliverable D7 - Event B language. Technical report, RODIN Consortium (2005) Available at http://rodin.cs.ncl.ac.uk/deliverables/rodinD7.pdf.
3. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Transactions on Software Engineering Methodology **11**(2) (2002) 256–290
4. Torlak, E., Jackson, D.: Kodkod: A relational model finder. In Grumberg, O., Huth, M., eds.: TACAS. Volume 4424 of LNCS., Springer (2007) 632–647
5. Mikhailov, L., Butler, M.: An approach to combining b and alloy. In: ZB'2002 – Formal Specification and Development in Z and B. Volume LNCS. (January 2002) 140–161
6. Leuschel, M., Butler, M.J.: ProB: A model checker for B. In Araki, K., Gnesi, S., Mandrioli, D., eds.: FME. Volume 2805 of LNCS., Springer (2003) 855–874
7. Ligot, O., Bendisposto, J., Leuschel, M.: Debugging Event-B Models using the ProB Disprover Plug-in. Proceedings of AFADL'07 (June 2007)
8. Matos, P.J., Marques-Silva, J.: Model checking event-b by encoding into alloy. Computing Research Repository **abs/0805.3256** (May 2008)

From ABZ to Cryptography (extended abstract)

Eerke A. Boiten

Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, UK.
Email: E.A.Boiten@kent.ac.uk

Abstract. This paper reports on work in applying ideas from the ABZ world to modern cryptographic protocols. It describes the important differences between this and more "traditional" application areas, and a number of promising approaches in formal methods.

Disclaimer

The nature of this paper is such that a bibliography giving decent coverage of the problems raised and attempted solutions from both sides of the fence would take up more than the total space available here – the reader is invited to look elsewhere, e.g. papers and research proposals at [5].

1 Natural Bedfellows?

At a first glance, cryptographic protocols provide exactly the kind of problems that formal methods are most suitable for and perform best at: short programs (most fit on a single page), based on rich algebraic mathematics, whose correctness is highly critical. However, the mathematics and the notions of security (correctness) are very different from the usual formal methods assortment.

2 Three Steps from the Ideal

Formal methods is about achieving correct systems. Ideally [12, 1], this correctness is achieved by construction: we use a "wide spectrum" language that encompasses both abstract specifications and executable programs, and transform one gradually into the other through small "correctness-preserving" steps. Refinement as a *process*, if you like, with the domain algebra, the properties of the problem, and a little creativity guiding us in creating a solution.

Slightly less desirable is post-hoc verification: proving that a proposed implementation is correct with respect to a specification (refinement as a *relation*), or that it satisfies certain properties. In the latter case, implementations and their properties may even be written in different languages.

If, given a specification and its intended solution, our mathematical framework does not help us in proving that it is correct, the next level is proof-checking. I.e., if someone comes along with a proof of correctness, we can formalise this, and then check mechanically that it discharges our overall proof obligation.

For modern cryptographic protocols (see below for what I mean by that), the state of the art is that proofs and proof methods are often insufficiently formalised for even proof checking to be a realistic prospective. So we are a full three steps away from the ideal way of achieving correctness.

3 Formal Methods and Cryptography

In the 1990s, formal methods techniques achieved major success in the modelling and analysis of cryptographic protocols, particularly work by the group using CSP around Oxford [9, 15] and by Paulson [13]. First, by considering non-deterministic choices of actions by the attacker, they allowed *abstraction* from attack *strategies* (and took anthropomorphism out of the equation: non-determinism encompasses “evil”). The second important aspect of this work was *automation*: using the theorem prover Isabelle in Paulson’s work, and using CASPER and the FDR[1] model checker in case of the Oxford group. However, this work was based on an abstraction of encryption which is an approximation. (Basically, the initial algebra assumption for encryption as the main constructor – implying an infinite algebra when all practical schemes work with fixed length bitstrings.) Thus, it may lead to false assurances of security. Also its emphasis on absolute notions of security does not sit well with modern cryptology.

4 Modern Cryptographic Protocols and Security

A modern cryptographic protocol may have the following properties:

- although its functionality is clear, its full set of desirable security properties may not be known yet;
- it contains explicit probabilistic elements, to mask input distributions and in “nonces”;
- its notion of security (correctness) is not an absolute one but approximate;
- moreover, this approximate correctness is relative to the computational resources available for an attack against it (which tends to imply an implicit probabilistic aspect);
- its security is not proved in an absolute sense but relative to the hardness of some computational problem;
- it uses primitives in a way which does not guarantee compositionality of the primitives’ properties.

All this means that the standard techniques and good intentions of formal methods do not work straight out of the box.

Many approaches to bridging the gap between formal methods and modern cryptography exist – see for example [4, 14, 7, 11, 8, 3]. These all have their advantages and disadvantages – but none are too close in spirit to the ABZ world.

5 What Do We Need, and What Has Been Done

Finally, I take a “bottom-up” view of how the ABZ world might approach the problem of “refinement for cryptographic protocols”: in which dimensions we would need to extend (say) standard Z states-and-operations refinement. This includes the following:

approximation Notions of correctness which are not exact but “close enough” – approximate refinement [6] would need to be strengthened to include fast convergence (“negligibility”). The cryptographic primitive of commitment, for example, requires two security properties – achieving both simultaneously is impossible, but schemes exist which approximate both with only negligible error.

probability Possibly protocols, and certainly attack models have a probabilistic element (“guessing”) to them. The work by McIver and Morgan [10] is a massive step forward in this area, and work on probabilistic refinement is continuing in several groups. Mingsheng Ying [16] has considered approximate probabilistic refinement.

action refinement Typical cryptographic protocols achieve a *single* objective through *multiple* communications between the parties involved. Thus, the granularity of actions decreases going from specification to implementation, requiring some kind of action refinement. Recent work by Banach and Schellhorn [2] is beginning to clarify issues of stutttering and upward vs. downward simulation in this area.

[1] The FDR tool is ©Formal Systems (Europe) Ltd.

attacks Protocols do not operate in isolation: multiple instances may run concurrently between different parties, and "dishonest" participants may not stick to the protocol. In the CSP work described above, this was modelled using non-deterministic choice over messages on a broadcast channel – is there an abstract data type analogue for this, and how do we model the limited ("polynomial") computing resources of such dishonest parties?

partwise and compositionality Refinement is monotonic with respect to most of the specification operators we use, allowing us to apply decomposition and partwise refinement. Approximation puts this under threat, and intuitively sensible notions of compositionality (e.g. [7]) have been shown to be unachievable for important cryptographic primitives.

All of this makes up a large research agenda to chip away at. Watch this space for a planned new EPSRC Network and new research in several of these areas.

References

1. R. Backhouse. *Program Construction: Calculating Implementations from Specifications.* Wiley, 2003.
2. R. Banach and G. Schellhorn. On the refinement of atomic actions. *ENTCS*, 201:3–30, 2008. Proceedings BCS-FACS Refinement Workshop 2007.
3. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426. Springer, 2006.
4. B. Blanchet and D. Pointcheval. Automated security proofs with sequences of games. In C. Dwork, editor, *CRYPTO'06*, volume 4117 of *Lecture Notes on Computer Science*, pages 537–554, Santa Barbara, CA, August 2006. Springer Verlag.
5. E.A. Boiten. Cryptography and formal methods project website. `www.cs.kent.ac.uk/~eab2/crypto/`
6. E.A. Boiten and J. Derrick. Formal program development with approximations. In H. Treharne, S. King, M. Henson, and S. Schneider, editors, *ZB 2005*, volume 3455 of *Lecture Notes in Computer Science*, pages 375–393. Springer, 2005.
7. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000.
8. A. Datta, A. Derek, J.C. Mitchell, V. Shmatikov, and M. Turuani. Probabilistic polynomial-time semantics for a protocol security logic. In *ICALP*, pages 16–29, 2005.
9. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.
10. A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer, 2004.
11. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In M. Naor, editor, *Theory of cryptography conference - Proceedings of TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151, Cambridge, MA, USA, February 2004. Springer.
12. C. C. Morgan. *Programming from Specifications.* International Series in Computer Science. Prentice Hall, 2nd edition, 1994.
13. L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.
14. B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *ACM Conference on Computer and Communications Security*, pages 245–254, 2000.
15. P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and A.W. Roscoe. *Modelling and Analysis of Security Protocols.* Addison-Wesley, 2001.
16. M. Ying. Reasoning about probabilistic sequential programs in a probabilistic logic. *Acta Informatica*, 39(5):315–389, 2003.

Exploiting the ASM method for Validation & Verification of Embedded Systems [*]

A. Gargantini[1], E. Riccobene[2], and P. Scandurra[2]

[1] DIIMM, Università di Bergamo, Italy
[2] DTI, Università di Milano, Italy

Abstract. We present the work in progress to build a model-driven environment for HW/SW co–design and analysis of embedded systems based on the UML, UML profiles for SystemC/multi-thread C, and the ASM formal method. The environment supports a model-driven design methodology which provides a graphical high-level representation of hardware and software components, and allows C/C++/SystemC code generation from models, a reverse engineering process from code to graphical UML models, and a transparent and tool-assisted system validation and verification based on ASMs.

SystemC (built upon C++) has emerged as de facto, open [3], industry-standard language for *system-level* models, specifically targeted at architectural, algorithmic, transaction-level modelling. Recently, a further improvement has been achieved by trying to combine SystemC with lightweight software modelling languages like UML (Unified Modeling Language) to describe system specifications. In accordance with the design principles of the OMG's *Model-driven architecture* (MDA), we defined a model-driven design methodology for embedded systems [5] based on the UML 2, a SystemC UML profile (for the HW side), and a multi-thread C UML profile (for the SW side), which allows modelling of the system at higher levels of abstraction (from a functional level down to RTL level).

In this paper, we present our work in progress on complementing this methodology with a *formal analysis process* for high level system validation and verification (V&V) which involves the Abstract State Machine (ASM) formal method.

Formal methods and analysis tools have been most often applied to low level hardware design. However, these techniques are not applicable to embedded system descriptions given in system-level design languages (like SystemC, SpecC, etc.) [7], since such languages are closer to concurrent software than to traditional hardware description. We propose to address the problem of formally analyze high-level UML-like embedded system descriptions by joining UML-like modelling languages with formal notations (like ASMs, Object-Z, Petri Nets, etc.) capable of eliminating ambiguities in the UML semantics, and then using techniques from *formal model analysis.*

[*] This work is supported in part by the project *Metodologie e tecniche model-driven per la progettazione e analisi di sistemi embedded tramite UML, ASM e SystemC* at STMicroelectronics.

Our overall goal is to provide a design environment where both the software application and the hardware architecture are described together by a multi-views UML model representing the mapping of the functionality (of the software application) onto an architecture. Moreover, through a formal analysis process the system components can be functionally validated and verified early at high levels of abstraction, and even in a transparent way (i.e. no strong skills and expertise on formal methods are required to the user) by the use of the ASM formal method and related tools.

The V&V framework

Figure 1 shows the general architecture of the proposed V&V framework. The environment consists of two major parts: a development kit (DK) with design and development components, and a runtime environment (RE) that is the SystemC execution engine. The DK consists of a UML2 *modeler* supporting the UML profile for SystemC and for multi-thread C[3], *translators* for forward/reverse engineering to/from C/C++/SystemC, and a *V&V toolset* based on the ASM formal method. The modeler is based on the Enterprise Architect (EA) UML tool by Sparx Systems and it is already used by our industrial partner [4].

The V&V toolset is built upon the ASMETA toolset [1]. Its essential components are depicted in Fig. 1 together with the phases (denoted in the figure with a number and a label) the designer (or analyst) undertakes in the analysis process. The process starts with the mapping (**phase 1**) of the SC-UML model of the system (exported from the EA-based modeler) into a corresponding ASM model (written in AsmetaL), which provides the basis for analysis. In practice, this transformation is effectively done by defining a set of semantic mapping rules between the SystemC UML profile and the AsmM metamodel. The `UML2AsmM` transformation can be completely automatized by means of a *transformation engine* such as the ATL engine developed within the Eclipse Modeling Project.

Once the ASM formal model of the system is generated, several phases can be executed in parallel. The basic simulation (**phase 2**) consists in simulating ASM models by AsmetaS. AsmetaS supports *axiom checking* (to check whether axioms expressed over the currently executed ASM model are satisfied or not), *consistent updates checking* for revealing inconsistent updates, *random simulation.* Scenario-based validation (**phase 3**) allows the designer to describe possible behaviours of the system as scenarios and test them within the AsmetaV validator. AsmetaV captures any violation in the required behaviour and reports some information about the coverage of the original model. Test-case generation (**phase 4**) allows to automatically derive test cases from the model of the system under test. The ATGT tool [2] is able to generate tests by exploiting the counter example generation of the model checker SPIN. They (and the validation scenarios) can be transformed in concrete SystemC test cases to test the conformance

[3] The methodology is fostered by a design process called UPES (Unified Process for Embedded Systems) [6] which extends the conventional Unified Process (UP), and by the UpSoC (Unified Process for SoC) for refining the HW platform model.

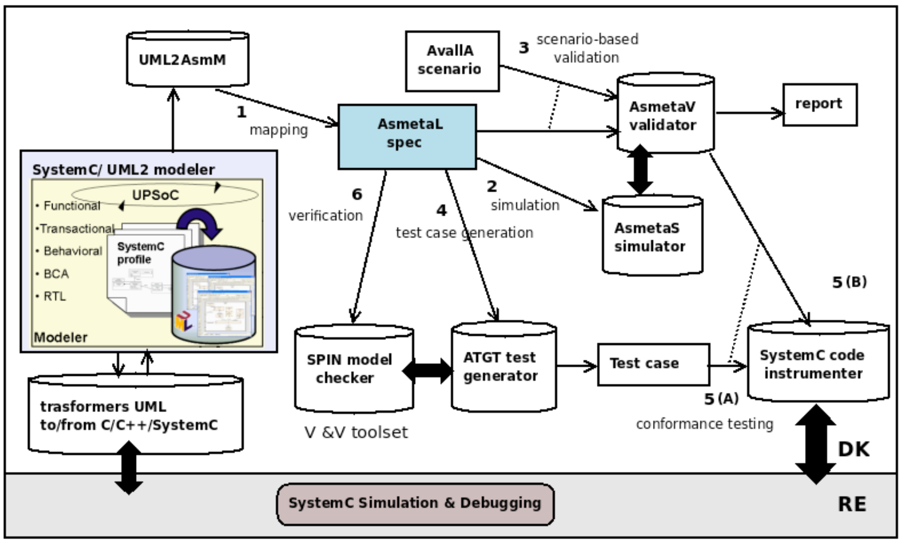


Fig. 1. V&V toolset

(**phases 5**) of the implementations with respect to their specification. We plan to support formal verification by model checking (**phase 6**). This requires transforming ASM models into inputs for model checkers, for example SPIN. In this case the user has to specify the desired properties (in linear temporal logic LTL).

We have been testing our analysis methodology on case studies taken from the standard SystemC distribution. Thanks to the ease in raising the abstraction level using ASMs, we believe our approach scales effectively to industrial systems.

References

1. The ASMETA toolset. `http://asmeta.sf.net/`, 2006.
2. A. Gargantini, E. Riccobene, and S. Rinzivillo. Using Spin to Generate Tests from ASM Specifications. In *10th Int. Workshop on Abstract State Machines*, LNCS 2589, p. 263-277. Springer, 2003.
3. The Open SystemC Initiative. `http://www.systemc.org`.
4. E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A model-driven design environment for embedded systems. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 915–918, New York, NY, USA, 2006. ACM.
5. E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A Model-driven co-design flow for Embedded Systems. *Advances in Design and Specification Languages for Embedded Systems (Best of FDL'06)*, 2007.
6. E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. Designing a unified process for embedded systems. In *Int. workshop on Model-based Methodologies for Pervasive and Embedded Software*. IEEE, 2007.
7. Moshe Y. Vardi. Formal techniques for systemc verification; position paper. In *DAC*, pages 188–192. IEEE, 2007.

BSmart: A Tool for the Development of Java Card Applications with the B Method

D. Déharbe[1], B. E. G. Gomes[1], and A. M. Moreira[1]

Federal University of Rio Grande do Norte; Natal, RN; Brazil
{david, bruno, anamaria}@consiste.dimap.ufrn.br

Abstract. BSmart is a tool to support a customized version of the B method for engineering Java Card software components for Smart Card applications, which require high-degrees of reliability and security.

1 Introduction

A smart card [1] is a portable computer device able to store data and execute commands in a highly secure way. Java Card [2] is a specialization of Java, providing vendor inter-operability for smart cards, and has now reached a *de facto* standard status in this industry. The strategic importance of this market and the requirement for a high reliability motivate the use of rigorous software development processes for smart card aware applications based on the Java Card technology. The B method [3] is a good candidate for such process, since it is a formal method with a successful record to address industrial-level software development.

In [4, 5], we proposed two versions of a Java Card software development method (called BSmart) based on the B method. Section 2 summarizes the main steps of the BSmart method.

In this paper, we present the current version of a tool (also called BSmart) to support the BSmart method. The BSmart tool must provide the automatable steps required by the method and some guidelines and *library machines* that can be useful during the development process. Further details are provided in Section 3. Related work and final considerations are drawn in Section 4.

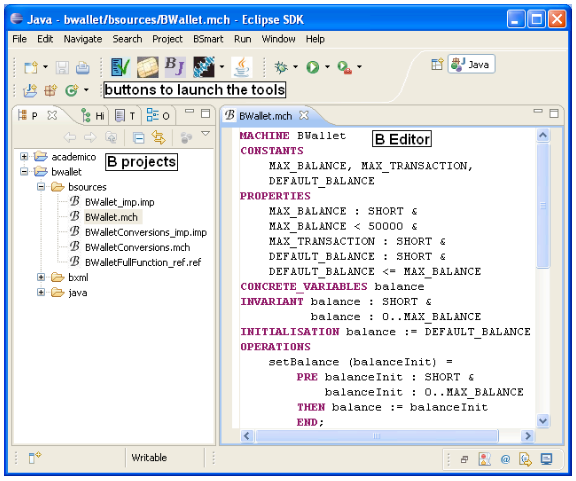
2 The BSmart Method

Smart card applications [1] have a client-server architecture, with the services being provided by the card, and the client (also called host application), executing in a terminal to which the card is temporarily connected. Communication is carried out through the APDU (Application Protocol Data Unit) protocol, defined in smart card standards [1]. The main feature of the BSmart method is to abstract, as much as possible, such platform particularities from the developers of smart card aware applications.

The card services specifier only needs to apply some refinement steps to his abstract (implementation platform independent) B specification (this specification is called `API.mch` in the following). These refinements have the goal of adapting the specification to Java Card standards and introducing platform specific aspects gradually. Also, as usual in the B method, other refinement steps may be needed to make the substitutions that define the operations directly translatable into Java Card code. Finally, the corresponding services provided by the card (Java Card code) will automatically be generated by the tool.

On the other side of the application, the developer of the host-side of the application will see a Java API, also generated by the tool, which corresponds to the interface of the original abstract specification from where the development started (`API.mch`). The client application code can then be developed in a completely platform independent way.

3 The BSmart Tool


Java - bwallet/bsources/BWallet.mch - Eclipse SDK
File Edit Navigate Search Project BSmart Run Window Help
buttons to launch the tools
Java
B projects
academico
bwallet
bsources
BWallet_imp.imp
BWallet.mch
BWalletConversions_imp.imp
BWalletConversions.mch
BWalletFullFunction_ref.ref
bxml
java
BWallet.mch
B Editor
Writable

```
MACHINE BWallet
CONSTANTS
    MAX_BALANCE, MAX_TRANSACTION,
    DEFAULT_BALANCE
PROPERTIES
    MAX_BALANCE : SHORT &
    MAX_BALANCE < 50000 &
    MAX_TRANSACTION : SHORT &
    DEFAULT_BALANCE : SHORT &
    DEFAULT_BALANCE <= MAX_BALANCE
CONCRETE_VARIABLES balance
INVARIANT balance : SHORT &
          balance : 0..MAX_BALANCE
INITIALISATION balance := DEFAULT_BALANCE
OPERATIONS
    setBalance (balanceInit) =
        PRE balanceInit : SHORT &
            balanceInit : 0..MAX_BALANCE
        THEN balance := balanceInit
        END;
```

Fig. 1. A snapshot of the BSmart interface.

The BSmart tool is an environment connecting several software components, each responsible for implementing a different step of the BSmart method. BSmart is being developed as an Eclipse IDE plugin. The choice for the use of Eclipse was made mainly because it allows a faster development of the user interface and the easier and faster distribution of the tool. Also, with this choice, we align BSmart with the RODIN platform, also developed in Eclipse.

The different components making up the BSmart environment are:

B parser and type checker jBTools [6] is the front-end of the environment. Responsible for parsing and type-checking, it generates an XML intermediary format that is later retrieved for further processing.

Proof obligations (PO) generator POs are generated with Batcave [7], that generates proof obligations in textual and Harvey [1] formats.

BSmart modules Generator This component generates a *full function* refinement, i.e., a refinement where all non-typing pre-conditions of each operation are checked for explicitly, and exceptions are raised whenever they are not satisfied. It provides a wizard for user-defined exception names or can generate them automatically. The tool includes the creation of a specific B machine to specify exceptions and a module, called *Conversions*, which defines useful abstraction for both client and server sides.

B-Java Card conformity checker This component verifies that some Java Card related restrictions (e.g., number of defined methods) are satisfied by the B specification.

B to Java Card code translator This component translates all the B implementation modules into Java Card programming code. It is derived from the existing Java

[1] http://harvey.loria.fr/

synthesis component available in jBTools [6] by the introduction of Java Card specific rules. The tool generates the Java Card server application as well as an API for the *host* side client. This API is responsible for abstracting the details of the Java Card framework classes being used to communicate with the *card* side server. It also handles coding the method arguments into communication protocol data units and decoding the corresponding results.

4 Related Work and Conclusions

Related work [8] and tools, such as AtelierB and the BToolkit, concerning the generation of imperative code from B specifications, have been around for a while. The generation of object oriented code or models is however still a matter of current research as in, e.g., [9, 10], and a Java code generation tool has been developed [6]. This tool is also a product of a smart card development project (*Project BOM*[2]), and takes care of some memory optimization issues. The development of BSmart builds upon this previous work. However, in this previous work, the generated code needs to be manually modified to incorporate the communication and codification aspects particular to the Java Card platform, whereas BSmart provides automated support to handle communication aspects.

At the moment of writing this paper, the definition of the method is in a mature stage, and our attention is now focused on the implementation of more robust versions of the BSmart tools and packaging them in a user-friendly environment. The integration of the AtelierB provers in Batcave and of a B animation tool is also planned for a next release of the tool.

References

1. Rankl, W., Effing, W.: Smart Card Handbook. John Wiley (2003)
2. Chen, Z.: Java Card Technology for Smart Cards: Architecture and Programmer's Guide. Addison Wesley (2000)
3. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge U. Press (1996)
4. Gomes, B., Moreira, A.M., Déharbe, D.: Developing Java Card applications with B. In: SBMF. (2005) 63–77
5. Deharbe, D., Gomes, B.G., Moreira, A.M.: Automation of Java Card component development using the B method. In: ICECCS, IEEE Comp. Soc. (2006) 259–268
6. Tatibouet, B., Requet, A., Voisinet, J., Hammad, A.: Java Card code generation from B specifications. In: ICFEM. Volume 2885 of LNCS. (2003) 306–318
7. Marinho, E., Jr, V.M., Tavares, C., Déharbe, D.: Batcave - um ambiente de Verificação Automática para o Método B. In: SBMF. (2007) 184–184
8. Bert, D., et al.: Adaptable translator of B specifications to embedded C programs. In: FME. Volume 2805 of LNCS. (2003) 94–113
9. Idani, A., Ledru, Y.: Object oriented concepts identifications from formal B specifications. In: FMICS. (2004)
10. Tatibouet, B., Hammad, A., Voisinet, J.C.: From abstract B specification to UML class diagrams. In: ISSPIT. (2002)

[2] `lifc.univ-fcomte.fr/RECHERCHE/TFC/rntl_bom.html`

Using Satisfiability Modulo Theories to Analyze Abstract State Machines

Margus Veanes[1] and Ando Saabas[2*]

[1] Microsoft Research, Redmond, WA, USA
margus@microsoft.com
[2] Institute of Cybernetics
Tallinn University of Technology, Tallinn, Estonia
ando@cs.ioc.ee

Abstract. We look at a fragment of ASMs used to model protocol-like aspects of software systems. Such models are used industrially as part of documentation and oracles in model-based testing of application-level network protocols. Correctness assumptions about the model are often expressed through state invariants. An important problem is to validate the model prior to its use as an oracle. We discuss a technique of using Satisfiability Modulo Theories or SMT to perform bounded reachability analysis of such models. We use the Z3 solver for our implementation and we use AsmL as the modeling language.

Protocols are abundant; we rely on the reliable sending and receiving of email, multimedia, and business data. But protocols, such as the Windows network file protocol SMB (Server Message Block), can be very complex and hard to get right. Model programs have proven to be a useful way to model the behavior of such protocols and it is an emerging practice in the software industry [6, 9, 11] to use model programs for documentation and behavioral specification of such protocols, so that different vendors understand the same protocol in the same way. The step semantics of model programs is based on the theory of ASMs [7] with a rich background universe [3]. This enables a range of ASM technologies [4] to be used for analysis of model programs. In the case of model programs, correctness assumptions about the model are often expressed through state invariants. It is important that the model is validated before it is used as a specification or an oracle. We describe a technique of using satisfiability modulo theories or SMT to perform bounded reachability analysis of a fragment of model programs. We use the SMT solver Z3 [5] and we use AsmL [8] as the modeling language. We extend the work in [10] through improved handling of quantifier elimination and extended support for background axioms, in particular bag or multi-set axioms.

The use of SMT solvers for automatic software analysis has recently been introduced [1] as an extension of SAT-based bounded model checking [2]. One advantage of the SMT approach is that it scales better for problems that depend on complex background theories, and the formula for which satisfiability is

* This work was done during an internship at Microsoft Research, Redmond.

checked is quantifier free, rather than propositional. The decision procedure for checking the satisfiability of the formula may use combinations of background theories. The formula is generated after preprocessing of the program. The preprocessing yields a normalized program where all loops have been eliminated by unwinding the loops up to a fixed bound. Unlike traditional sequential programs, model programs operate on a more abstract level and often make use of comprehensions. Moreover, model programs use parallel updates and rich background data structures like sets, maps and bags.

A model program is a finite collection of basic ASMs indexed by actions. The following model program, called *Credits*, is an example of a model program written in AsmL. It specifies how a client and a server need to use message ids, based on a sliding window protocol. Here the client sends requests to the server and the server sends responses back to the client.

```
var window as Set of Integer = {0}
var maxId as Integer = 0
var requests as Map of Integer to Integer = {->}

[Action]
Req(m as Integer, c as Integer)
  require m in window and c > 0
  requests := Add(requests,m,c)
  window := window difference {m}

[Action]
Res(m as Integer, c as Integer)
   require m in requests
   require requests(m) >= c
   require c >= 0
   //require requests.Size > 1 or window <> {} or c > 0 <-- bug
   window := window union {maxId + i | i in {1..c}}
   requests := RemoveAt(requests,m)
   maxId := maxId + c

[Invariant]
ClientHasEnoughCredits()
   require requests = {->} implies window <> {}
```

The *Credits* model program illustrates a typical use of model programs as protocol-specifications. Actions use parameters, maps and sets are used as state variables and a comprehension expression is used to compute a set. Each action has a guard and an update rule given by a basic ASM. For example, the guard of the `Req` action requires that the id of the message is in the current window of available ids and that the number of credits that the client requests from the server is positive. The state invariant associated with the model program is that the client must not starve, i.e. there should always be a message id available at some point, so that the client can issue new requests.

There is a mistake in the model indicated by the missing require-statement. There is a two-action trace leading to a state where the invariant is violated due to this, e.g. the trace `Req(0,1),Res(0,0)`.

There are several different ways of how model programs can be checked for invariant violations. One way is to do explicit state exploration and to use model checking techniques, e.g. this is supported in Spec Explorer [11]. Another approach, that does not require action parameter domains to be provided up-front, is to represent the bounded reachability problem of the negated invariant as a formula and to check for its satisfiability using a theorem prover.

The *bounded reachability formula* for a given model program P, step bound k and reachability condition φ is:

$$Reach(P, \varphi, k) \quad \overset{\text{def}}{=} \quad I_P \wedge (\bigwedge_{0 \le i < k} P[i]) \wedge (\bigvee_{0 \le i \le k} \varphi[i]) \tag{1}$$

where I_P is the initial state condition, $P[i]$ is a formula describing step i of the model program, which is an application of some enabled action from the i'th state, and $\varphi[i]$ is φ in the i'th state. The reachability condition φ is typically the negated state invariant. If $Reach(P, \varphi, k)$ is satisfiable, its model can be used to extract an action trace that leads from the initial state to a state violating the invariant. The formula $Reach(P, \varphi, k)$ is typically quantifier free, but involves the use of background theories such as arithmetic, set and multi-set axioms, and map axioms, which makes the use of an SMT solver such as Z3 possible for this kind of analysis.

References

1. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. In A. Valmari, editor, *SPIN*, volume 3925 of *LNCS*, pages 146–162. Springer, 2006.
2. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.
3. A. Blass and Y. Gurevich. Background, reserve, and Gandy machines. In *Proceedings of the 14th Annual Conference of the EACSL on Computer Science Logic*, pages 1–17. Springer, 2000.
4. E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003.
5. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'08)*, LNCS. Springer, 2008.
6. W. Grieskamp, D. MacDonald, N. Kicillof, A. Nandan, K. Stobie, and F. Wurden. Model-based quality assurance of windows protocol documentation. In *First International Conference on Software Testing, Verification and Validation, ICST*, Lillehammer, Norway, April 2008.
7. Y. Gurevich. *Specification and Validation Methods*, chapter Evolving Algebras 1993: Lipari Guide, pages 9–36. Oxford University Press, 1995.
8. Y. Gurevich, B. Rossman, and W. Schulte. Semantic essence of AsmL. *Theor. Comput. Sci.*, 343(3):370–412, 2005.
9. J. Jacky, M. Veanes, C. Campbell, and W. Schulte. *Model-based Software Testing and Analysis with C#*. Cambridge University Press, 2008.
10. M. Veanes, N. Bjørner, and A. Raschke. An SMT approach to bounded reachability analysis of model programs. In *FORTE'08*, LNCS. Springer, 2008.
11. M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson. Model-based testing of object-oriented reactive systems with Spec Explorer. In R. Hierons, J. Bowen, and M. Harman, editors, *Formal Methods and Testing*, volume 4949 of *LNCS*, pages 39–76. Springer, 2008.

DIR 41 Case Study
How Event-B Can Improve an Industrial System Specification

Christophe Metayer[1] and Mathieu Clabaut[2]

[1] Systerel `christophe.metayer@systerel.fr`
[2] Systerel `mathieu.clabaut@systerel.fr`

Abstract. A case study has been carried to verify the specification of an automated interlocking system by means of system modeling with event-B. This paper gives an overview of the work done and of some of the results.
This study was supported by the RATP[3].

1 Context and Goals

Some RATP units are responsible for evolution and maintenance of an automated interlocking specification document.

In order to improve their paper and pencil process, RATP asked Systerel if Event-B could be useful to them. An eight month study was launched whose main goal was to help RATP improving their confidence in their interlocking specification, by applying an Event-B approach on rewriting their requirement document.

2 Description of Work

The main difficulty that arose at the very beginning of the study was to answer the initial question: *what does a "working interlocking system" mean?*

In an attempt to both answer this question and stay within the expectations of the railway's people, we chose to allocate a great chunk of time to the elaboration of our own requirements specification, by rewriting the RATP specification with the organization and the wording of our choice. This task was carried before doing any modeling work and was aimed at introducing safety principles and safety concepts step by step.

The obtained requirement specification was to be approved by the domain experts and to serve as a reference for the modeling task.

Our modeling process was thus broken down as follow:

1. Writing an autonomous requirements specification approved by the domain experts,

[3] French organization in charge of Paris transportation.

2. Designing a refinement plan,
3. Modeling the system and proving it correct.

The whole process was highly iterative. Indeed, it often appears while modeling that some concepts still need refinements or adjustments which in turn lead to a rework of the refinement plan or the model.

Table 1. Structure of the B model.

Collision avoidance (2 levels)
Environment structures (5 levels)
Control principles (4 levels)
Instantiation (4 levels) - signal - switch command - transit locking - opposing locking

The achieved model, which overall structure is shown in Table 1, contains fifteen levels of refinement.

3 Results

The study led to several interesting results either by giving some clues about a better way for building an event-B model for a real world system, or about the quality of the automated interlocking specification, object of this study.

3.1 Organization

Our first intention was to teach some basics of event-B to the domain experts in order to allow them to at least follow, and at best approve, the modeling steps. It proves as a bit utopian. It would certainly have needed a lot more of teaching effort to be efficient.

It is of interest to note that at the end of the study, the results were perceived by the domain experts as *magical* ones: the failure scenarios exhibited seem to come from nowhere (in fact, proof obligations of the B model that couldn't be discharged), in contrast to the scenario they are used to, which come from several decades of field feedbacks. Those *magical scenario* nonetheless were pertinents.

3.2 Modeling Process

The modeling process we have planned had some place for improvements (some of which were applied during the study). They are summed up hereafter:

Process for Industrial Strength Models There is a need for a more robust modeling process, perhaps somewhat similar to those being used at the software level. An industrial modeling process probably needs the following phases:

- Requirements specification,
- Validation tests definition,
- Refinement planning,
- Modeling and proving,
- Validating.

The requirements specification rewriting phase done at the beginning of the study involved a lot of refactoring and lead to a final document that was too far from the domain experts expectations to be fully approved. Animation looks like *a must have* to ease model validation by domain experts.

It is also worth to note that seeking for a posteriori justification, as was done in this study, is very difficult. Indeed, the system was obviously not designed to be proved. In fact, it appears to us that most of existing industrial systems were not designed with validation in mind.

Modeling Techniques Our model didn't explicitly take into account degraded cases for the system. It proved to be a bad design choice and it was a burden for the proof of the model, as event guards become more and more complex throughout the refinement levels. As a consequence, it appears that models need to be totally closed, and thus should not only take into account the controller and its environment, but also degraded cases.

As a conclusion, we can also say that refinement appears more and more as an essential concept for successful modeling.

3.3 DIR 41 Analysis

Some essential concepts for the railway domain were refined during the study. For example the *safety statement* was refined to a *safety preservation* one: the interlocking system actions should not decrease the current safety level.

And probably the most prominent result is that four potential safety flaws were exhibited (with an expected low probability of occurrence), which are now being tackled by the RATP teams. It also revealed the existence of several implicit hypotheses on the environment behaviour or on the design of the railway network.

4 Conclusion

Modeling a system with event B proved to be very interesting for pointing out potential safety flaws and for proving global safety.

This way of modeling allows a B expert with little knowledge in an industrial domain to quickly grasp the domain core concepts. It is still very difficult to involve the domain experts in the whole process and we have high expectations that model animation would improve this.

Formalizing Delegation in Role-Based Access Control Models

Hassan Takabi, and Ali E. Abdallah

E-Security Research Centre
London South Bank University
103 Borough Road
London SE1 0AA, UK
{takabih, abdallae}@lsbu.ac.uk

Abstract. Role-Based Access Control (RBAC) is a high level authorization model in which access decisions are based on the roles that users hold within an organization. RBAC has proved to be very useful, particularly for large organizations, because it offers scalability, consistency and ease of maintenance. Delegation is an important concept in authorization which is often deployed in most real access control systems. In RBAC, delegation essentially means the ability of a user who occupies a certain role to authorize another user to perform the tasks permitted by that role. However, there are several informal variations to this idea which, when formalized, result in various semantically different RBAC delegation models. In this paper, we clarify the key role-based delegation concepts and define a number of RBAC delegation models with different characteristics. We start by introducing delegation to the simplest core RBAC model. We refine the core RBAC model to support temporal role-base access control and show how to integrate delegation and revocation in the temporal model. These models are formulated in the specification notation **Z**. The semantics of each model is given by defining the relevant access control monitor which takes a subject and a task and determines whether the request should be granted or denied.

Keywords: Delegation, Role-Based Access Control, Authorization, Formal Models, **Z** specification.

Formal Requirements Specification for Railway Signaling Systems

Miyoung Kang[1], Dae-Yon Hwang[1], Junkil Park[1], Jin-Young Choi[1], Jong-Gju Hwang[2]

[1]Dept. of Computer Science and Engineering, Korea University, Seoul, Korea
{mykang, dyhwang, jkpark, choi} @formal.korea.ac.kr
[2]Korea Railroad Research Institute
jghwang@krri.re.kr

Abstract. A project to develop a railway technology for the future is currently underway in Korea. This paper introduces the research which describes the formal requirement specifications for obtaining safety within the railway signaling systems. The railway signaling systems are being changed to a computer based system from the previous hardwired based system, because current railway systems are required to be more complicated structure. Especially, several issues about reliability and safety of the railway system are being identified. To solve the problems of the reliability and the safety, we are using the formal specification language Z in the early development stage. It can be increasing the safety and reliability of the railway system.

Keywords: Railway signaling systems, Formal requirements specification, Z

1 Introduction

A project to develop a railway technology for the future is currently underway in Korea. In particular, Korea Railroad Research Institute(KRRI) is performing national project assessing safety of the railway signaling systems in Korea, and is developing an accident prevention technology. There are six projects led by KRRI. One of the project is study on the formal requirement specification. This project introduces the research which describes the formal requirement specifications for obtaining safety within the railway signaling systems.

Railway signaling systems is the safety critical system that controls the route and speed of the train. If this system malfunctions during operation, it could result in catastrophic materialistic and human damages. In Korea, the railway signaling systems recently switched to a computer based system from the preexisting hardwired based system, which caused the system to adopt a more complicated structure. As a result, the issues regarding the reliability and safety of the system are constantly appearing on the social agenda. In order to resolve these issues, the international standard such as IEC and IEEE are recommending the standardization of the system during its development process as well as the standard proposal which will enable the documents to be created in a correct method, beginning from the step in which requirements are specified in detail [1]. The purpose of the project is to establish requirements specification suitable for Korean conditions based on international standard, and to add formal methods[2] to given requirement specification. This can be an example of practical specification. We use Statechart and Z[3] as formal specification methods and here we are going to explain how we specify the system using Z. Our Z specification consists of 15 schemas which are simple examples.

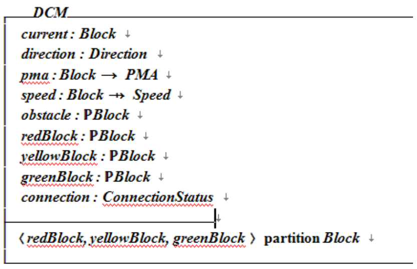
2 Mock-up for the future railway technology

Instead of the actual system, KRRI has provide us with a mock-up system, for us to conduct research on the applicability of the technology for safety assessment. This part of the project is to describe Distance Control Module(DCM) and Automatic Train Protection(ATP) of the mock-up system using Z notation. The requirements specification in the natural language is as follows.

In the train moves according to the Permissive Movement Authority(PMA) of each block, and the PMA message is created by DCM, and is sent to the ATP. The main functions of DCM are as follows. First, controlling the distance between trains: DCM must maintain a minimum distance between trains even within the worst malfunctioning scenario for safety reasons. Second, confirming the location of the train: DCM confirms the location of the train by constantly communicating with the ATP. Third, observing the movement and direction of the train: DCM transmits the operation information (movement direction) of the train to the ATP. Then, DCM compares the newly received current location of the train with the preexisting date stored in DCM, and then check whether or not the train is progressing according to the plan. Fourth, processing the temporary speed limit commands: DCM sets a temporary speed limit when the train passes construction areas, maintenance areas, windy areas, and areas that necessitate a decrease in speed. Finally, DCM can open and close a block under the following two circumstances. Relevant blocks are closed in a circumstance in which ATS simulator has transmitted a closing command of a particular block to DCM, and also in an emergency situation in which trains have been lost.

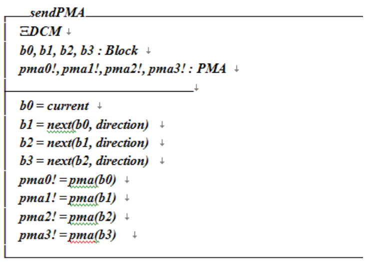
3 Z Specification of the DCM

Z Specification descriptions are divided into 3 parts: Data model, state model, and safety model. DCM schema and sendPMA schema are shown as examples of state model.



```
DCM
current : Block
direction : Direction
pma : Block → PMA
speed : Block ⇸ Speed
obstacle : ℙBlock
redBlock : ℙBlock
yellowBlock : ℙBlock
greenBlock : ℙBlock
connection : ConnectionStatus
────────────
⟨redBlock, yellowBlock, greenBlock⟩ partition Block
```

Fig. 1. DCM



```
sendPMA
ΞDCM
b0, b1, b2, b3 : Block
pma0!, pma1!, pma2!, pma3! : PMA
────────────
b0 = current
b1 = next(b0, direction)
b2 = next(b1, direction)
b3 = next(b2, direction)
pma0! = pma(b0)
pma1! = pma(b1)
pma2! = pma(b2)
pma3! = pma(b3)
```

Fig. 2. sendPMA

Fig.1. specifies the state of the basic information that DCM needs to have. DCM contains such information as blocks occupied by train now, driving direction of train, PMA setting of all blocks(every block has to have one PMA setting among RED, YELLOW, or GREEN), blocks set as obstacle, and communication connection status between DCM and ATP. Fig.2. sendPMA transfers information including from current location of the train to the fourth block which has occupied block of train driving direction to ATP.

4 Conclusion

The development of a railway technology for the future is currently underway in Korea. As part of the development our project goal is to promote the use of formal methods on requirements engineering of the future Korean railway technology to increase the safety and reliability of the railway signaling systems. In this research, a way of creating the requirement specification method that is in accordance with the international standard is studied, and application of the definite methods suitable for the railway signaling systems is investigated. Also, based on this research, we are developing examples of requirements specifications for particular functions of the mock-up system, which can be applied to the actual industrial sites. This paper describes our research to increase the safety and reliability by using the requirement specifications in Z notation.

References

1. R. Bell, "Introduction to IEC 61508", Conference in Research and Practice in Information Technology, Vol. No.55, 2005
2. E. Clarke and J. Wing, "Formal Methods: State of the Art and Future Directions", *ACM Computing Surveys*, vol. 28, No.4, pp.626-643, 1996
3. J.M.Spivey, The Z Reference Manual, 2nd ed, Prentice Hall, Englewood Cliffs, NJ, 1987