Feature Composition – Towards product lines of Event-B models

Ali Gondal, Michael Poppleton, Colin Snook

Dependable Systems and Software Engineering Group
University of Southampton, Southampton SO17 1BJ, UK
{aag07r, mrp, cfs}@ecs.soton.ac.uk

Abstract. Event-B is a formal language for modelling reactive systems, based on set theory and first-order logic. The RODIN toolkit provides comprehensive tool support for modelling and refinement in Event-B, analysis and verification using animator/model-checkers and theorem provers. We consider the need to support reuse, in particular product line reuse, in such a formal development method.
Feature modelling is an established technique for reuse in product lines. We introduce concepts of feature modelling and composition in Event-B to support the reuse of formal models and developments. A prototype feature composition tool has been developed (as a RODIN plugin) for Event-B, based on the Eclipse Modelling Framework (EMF). Using an MDD philosophy, the tool extends the Event-B language metamodel to a composition metamodel, and implements prototype composition patterns for Event-B features. Thus, a required composite model can be constructed by selecting, specializing, and composing input features in a defined way. The tool is the first step towards full feature modelling for product line model reuse for Event-B. We describe future work required to meet this goal.

1 Introduction

Formal Methods provide mathematically based languages, tools and techniques for specifying and verifying systems during construction. They allow identification of inconsistencies, ambiguities and defects earlier in the software development life-cycle and reduce the need for unit and integration testing [1]. Verification conditions called *proof obligations* (POs) take the form of mathematical theorems which state correctness properties of models and their refinements. Successful application of formal methods can be seen in aerospace, transportation, defence and medical sectors [1, 2]. Improvements in formal specification languages, verification techniques and robust tools are ongoing, in particular, by the DEPLOY[1] and RODIN[2] projects, including industrial partners such as Bosch,

[1] DEPLOY - Industrial deployment of system engineering methods EU Project IST-214158. http://www.deploy-project.eu

[2] RODIN - Rigorous Open Development Environment for Open Systems: EU Project IST-511599. http://rodin.cs.ncl.ac.uk

SAP, and Siemens and Space Systems Finland. These projects are developing tools, as well as strengthening the theoretical base, for the formal specification language Event-B [3].

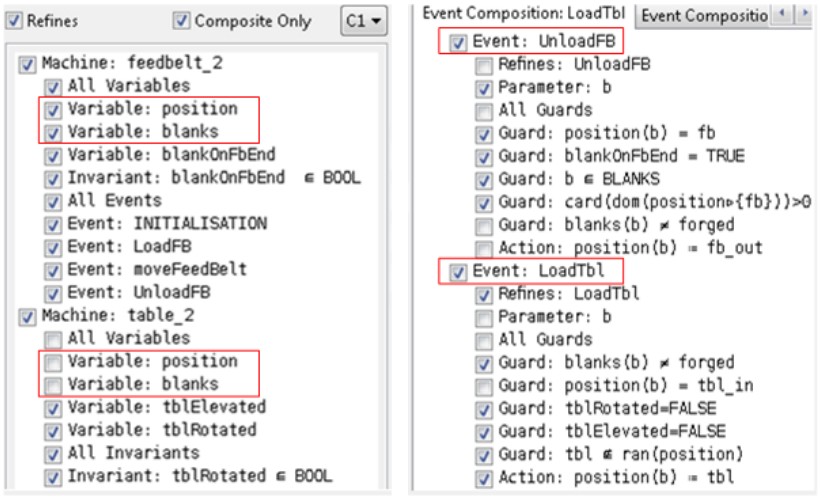
Event-B, based on set theory and first-order logic, is used for modelling and analysing discrete event systems and provides built-in generation and verification of proof obligations. It is a successor of Abrial's B language [4], developed in the RODIN and earlier EU projects. After completion of the RODIN project, the DEPLOY project is now deploying this work into industry. The RODIN toolkit [5] provides support for modelling, animation, model-checking and proof using Event-B. It is an Eclipse based IDE, and easily extensible. Refinement is the core development process of introducing more details in each step from abstract specification to the concrete implementation model in Event-B. Any refined model must be proved to be a true refinement of the abstract model.

A software product line (SPL) refers to a set of related products having a common base and built from a shared set of resources [6]. SPLs focus on the problem of software reuse by providing automated ways to build families of software products sharing commonalities, and differing by variabilities of structure. Feature modelling is a model-driven approach which gives a means to define commonalities and variabilities in terms of atomic requirements *features*. Several tools have been developed for supporting feature modelling for SPL engineering [7]. For SPLs of critical systems, there is a need for the automated verification that formal methods provide. This paper discusses our approach to introduce product line reuse in Event-B using feature modelling concepts and reports on tooling developments towards the full Feature Modelling Tool (FMT) for modelling of SPLs in Event-B. The example used in the paper is discussed in section 2. Section 3 gives a technical overview of the Event-B language. The notion of feature modelling in Event-B is discussed in section 4 followed by the tool discussion, related and future works in sections 5, 6 and 7 respectively.

2 Feature Composition Example

This example has been taken from the Production Cell [8] case study which we have modelled in Event-B. This is a reactive system which has been modelled in a number of formalisms. The purpose of the example here is to demonstrate the prototype tool that we have developed rather than its Event-B modelling. There were multiple features in Production Cell model but we only consider two features here for brevity, i.e. feed belt and table. Metal blanks enter the system through the feed belt and are dropped one by one on to the table from where other components such as a robot may pick them up and deliver them to another component for further processing. It happens in this example that features map to physical components of the system, e.g. table, robot etc. We chose this example to represent a product line of Production Cell systems, where different feature configurations result in different instances of a Production Cell. Variabilities are the number and connectivity of features, e.g. we can build a system with

multiple robots and belts to increase throughput. All the features are modelled generically for use in various configurations, and are verified separately.



(a) Feature Composition (b) Sub-Feature Composition

Fig. 1. Feature Composition Editor

3 Event-B Language

An Event-B model consists of a *machine* - modelling dynamic data and behaviour - and zero or more *contexts* - modelling static data structures or configurations. This separation of behaviour allows the use of different contexts to parametrize the machines. Note that there is no concept of modularization in Event-B, so a model represents a complete system at a particular level of refinement. The state transition mechanism over the machine's *variables* is given by the *event*, which comprises *parameters* (also called arguments or local variables), *guards* and *actions*. The guard is a condition on the event parameters and machine variables that defines the enabledness of the event: the event is only enabled when all of its guards are true. The action is the update operation on a state variable. The syntax of an event e with guards G, variables v and actions A is: $e =$ `when` $G(v)$ `then` $A(v)$. Examples of two events (UnloadFB & LoadTbl) can be seen on Fig. 1(b). The *invariant* is a state predicate specifying correctness properties that must always hold. Variables are typed by invariants. Many POs concern invariant preservation, i.e. correctness of the system is defined and preserved through the invariants. All events must preserve invariants and any violation of invariants will lead to the system being inconsistent. An `INITIALIZATION` event is used to specify the initial values for the variables.

The context (static data) in Event-B contains *sets*, *constants*, *axioms* and *theorems*. Sets are used to define the types and axioms describe the properties of the constants. Theorems must be proved to follow from axioms.

Event-B provides support for refinement where structural and algorithmic detail can be added during each refinement step; new events can be added and existing events can be extended. Variables may be added or transformed. Refinement will usually reduce non-determinism. Similarly, contexts can also be extended to add more details to the model. Refinement proof obligations are generated by the tool to verify that a refined model is a correct refinement of the abstract model. An EMF [9] metamodel for Event-B has been developed as part of the DEPLOY project.

4 Feature Modelling in Event-B

Feature modelling is a well-known technique for reuse in product lines. Methodologically speaking, for its application in an Event-B setting, some form of domain engineering activity comes first. At very least, an instance of the product line should be fully developed, followed by the engineering of a variant system or two. Commonality/variability analysis should be undertaken and followed by the incremental building of a domain feature model and database. Work is under way developing this methodological and domain engineering work, which will be reported elsewhere.

In this paper we introduce a prototype feature composition tool as an initial step towards the development of a more comprehensive tooling for feature modelling. For such a full Feature Modelling Tool (FMT), we will need to define a metamodel for the feature modelling language. The FMT will consist of a feature model editor and a feature instance editor. The feature model editor will be used to build the feature models for different product families and the feature instance editor will provide a configuration mechanism for choosing various features to instantiate a new product line member, somewhat similar to configuration diagram of FeaturePlugin [10]. The feature model will also specify any constraints needed to maintain the correct selection of features in a particular instance. Feature composition rules will respect these constraints. The instance editor will use the feature composition tool described in this paper to compose selected features to build the instance system. We are planning to develop the instance editor in such a way that it can produce an instance using the composition descriptions defined in the composition meta-language as discussed in our earlier work [11]. This framework consists of two layers of metamodelling where a feature model will conform to the feature metamodel and at the same time will serve as a metamodel for the instance model. Validation criteria will be needed to verify the correct instantiation of the metamodels at each level.

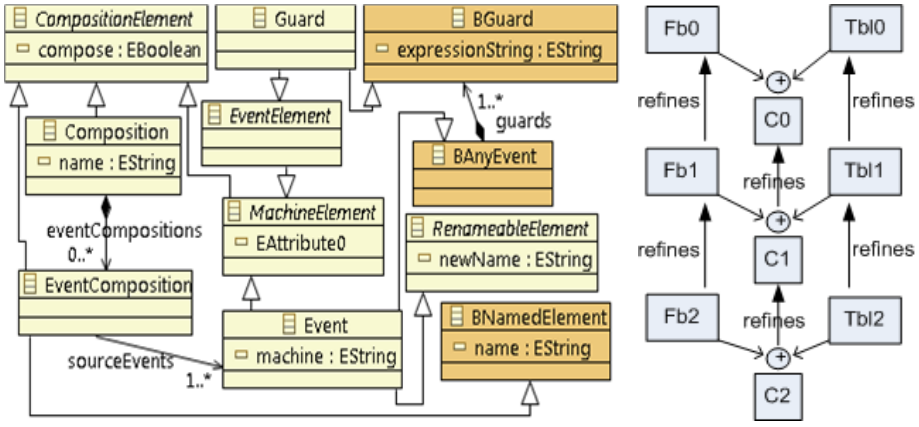
The feature has been defined as "a logical unit of behaviour specified by a set of functional and non-functional requirements" [12]. We define the concepts of "feature" and "sub-feature" in Event-B as atomic units of reuse, specialization and composition. This is in order to preserve the semantics of Event-B and to

formally verify the product-line members. A *feature* is thus a small, coherent and syntactically complete Event-B model which consists of a machine and zero or more seen contexts. This allows a feature and its refinements to be verified using the RODIN provers. A notion of *sub-feature* may be useful: when a feature cannot be reused as a whole, we might be interested in reusing some parts of a feature. Thus, a sub-feature is part of a feature which is syntactically incomplete but can be reused when composed with other sub-features, e.g. in Event-B, an event or a variable with its associated invariant(s) can constitute a sub-feature. The following is our definition of feature and sub-feature in BNF[3].

$$
\begin{aligned}
&Feature ::= Context \mid Machine\ Context^{+} \\
&Machine ::= Name\ Variable^{+}\ Invariant^{+}\ Theorem^{*}\ [Variant] \\
&\qquad\qquad \texttt{INITIALIZATION}\ Init\ Event^{+} \\
&Context ::= Name\ \{Set^{+}\ Axiom^{*}\ Theorem^{*} \mid \\
&\qquad\qquad\qquad Set^{*}\ Constant^{+}\ Axiom^{+}\ Theorem^{*}\} \\
&SubFeature ::= EventSF \mid VariableSF \mid InvariantSF \mid ContextSF \\
&EventSF ::= \texttt{INITIALIZATION}\ Init \mid Event \\
&VariableSF ::= Variable^{+}\ InvariantSF \\
&InvariantSF ::= Invariant^{+} \\
&TheoremSF ::= Theorem^{+} \\
&ContextSF ::= Set^{*} \mid Constant^{*} \mid Axiom^{*} \mid Theorem^{*}
\end{aligned}
$$

We start system modelling in Event-B by defining the features at an abstract level and then refine these features gradually by adding further detail at each refinement step. Splitting requirements across features and then modelling and refining feature-wise, separates concerns and should reduce complexity - application experience will tell. We call each feature and its chain of refinements a *feature development*. Developing earlier work [11, 13], we regard these feature developments as units of reuse. They will be specialized - by addition or alteration of information - and composed in various ways in the process of assembling (instantiating) an instance system in the target product line. For example, two *EventSFs* are composed after specialization and conflict resolution into a single *EventSF* named 'LoadTbl' during the composition of two Features i.e. feedbelt_2 & table_2, as shown in Fig. 1. The processes of specialization and composition will occur in general at all refinement levels of the input feature developments. Note that the relation between requirement feature and its implementation is more complex than one to one and feature composition through non-linear refinements will layer in complex, possibly cross-cutting, structure.

[3] The syntax used is: [ ] means optional (0 or 1), { }* means 0 or more occurrences, { }$^{+}$ means 1 or more occurrences and | means OR.



(a) Part of Composition Metamodel (b) Example Structure

Fig. 2.

5 Feature Composition Tool

An initial version of the feature composition tool is available as a plugin to the RODIN platform[4] [5]. It provides a simple structured cut-and-paste composition of features and guides the user in identifying and resolving any conflicts. We are currently working on a number of composition/specialization patterns to maximize automation while not restricting user expressiveness.

The plugin was developed in Eclipse using Java in a model-driven approach. We built a metamodel for the composition model which inherits from the Event-B metamodel. We then used EMF [9] to generate the code from the composition metamodel. The major advantage is in the ease of extension through the metamodel where the code is automatically generated by EMF and then customized. Fig. 2(a) shows part of the composition metamodel for event composition where Event, Guard and EventComposition inherit from Event-B metamodel, i.e. BAnyEvent, BGuard and BNamedElement respectively. Event inherits 'newName' from RenameableElement which allows a new name to be given to an Event, e.g. to resolve naming conflicts. The collection 'sourceEvents' represents the set of events being merged together to form a single 'EventComposition' which inherits 'name' attribute from BNamedElement. All composition elements inherit a 'compose' boolean which represents the selection of that element for composition.

The tool provides a composition editor[5](Fig. 1) to select the features (in a similar manner to existing feature modelling tools, e.g. [10]) that need to be

[4] See `http://sourceforge.net/projects/rodin-b-sharp/` and wiki entry at `http://wiki.event-b.org/index.php/Feature_Composition_Plugin`

[5] The earlier prototype was contributed by Christopher Franklin (a University of Southampton Intern)

composed resulting in the composition model. This model is then used by the tool to transform/compose the input features into a composite Event-B feature. The composition model is serialized in the RODIN model repository for replaying the composition later. The editor provides conflict resolution functionality and highlights any conflicts such as multiple declarations of *variables* or *events* with the same name in different input features. It provides facilities for making the input models disjoint before composition to automatically resolve any conflicting element names. It can also automatically resolve conflicting elements by deselecting the repeating/redundant information in different models (see variables 'position' and 'blanks' on Fig. 1(a)). The editor also provides an option for composing sub-features such as *events* (Fig. 1(b) shows the composition of two events). The tool is also capable of composing features at different refinement levels. The composite feature is a typical Event-B model and is automatically checked by the RODIN static checker for any errors. Similarly, proof obligations (POs) for the composite model are generated in normal fashion and the RODIN provers discharge any POs automatically if they can. Fig. 2(b) shows the structure of our example features refined up to two levels and then composed using the tool. The composition of features at each level needed some extra invariants, guards and merging of events. Fig. 1(b) shows the composition of two events into one and de-selection of redundant elements to resolve conflicts.

6 Related Work

To our knowledge, there is no tool support for feature modelling within the formal methods domain, although, there are tools which support feature modelling for product lines such as XFeature[6], FeaturePlugin [10] etc. The underlying concepts discussed in [14] are quite similar to what we want to achieve in the domain of formal product line development. Another area that is closely related to our work is the definition of composition patterns. These patterns will enable us to write composition rules when composing Event-B features. The RODIN toolkit is already in the phase of adopting the model transformation and code generation facilities of EMF[7].

7 Conclusion & Future Work

We have given an overview of our approach to introduce the concepts of feature modelling within formal methods using Event-B. Our prototype feature composition tool is an initial step towards the development of a feature modelling tool for configuring Event-B features and composing them to instantiate software product line systems by reusing and extending existing features as mentioned in section 4. This is required because existing feature modelling tools don't provide

[6] Feature Modelling Tool `http://www.pnp-software.com/XFeature/Home.html`
[7] See `http://wiki.event-b.org/index.php/EMF_framework_for_Event-B`

enough facilities to give semantics to features and the resulting formal verification capabilities such as offered by Event-B. Our prototype tool will enable us to experiment with different case studies and to improve the tool requirements and underlying notations for feature modelling. This remains work in progress.

The example of section 2 revealed additional tool requirements and research questions.We will require a composition management capability to record, manage and replay the sequences of compositions and specializations of features required by a target system. This should substantially increase productivity. Another area to explore is reusing proofs. When we compose Event-B features into a composite feature, the tool generates proof obligations (POs) for consistency and refinement checking. Most of the POs for the input features still exist for the composite feature, and may have already been discharged interactively. Hence, it would be useful if the tool could reuse interactively discharged POs to save user time and effort. This might be achieved through the composition of proof trees while composing their associated features.

References

1. Abrial, J.R.: Formal methods in industry: Achievements, problems, future. In: ICSE '06: Proceedings of the 28th ICSE, NY, USA, ACM (2008) 761–768
2. Bowen, J.P., Hinchey, M.G.: The use of industrial-strength formal methods. In: COMPSAC '97, Washington, DC, USA, IEEE Computer Society (1997) 332–337
3. Metayer, C., Abrial, J.R., Voisin, L.: Event-B language. Rodin deliverable 3.2, EU Project IST-511599 -RODIN (May 2005)
4. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, New York, NY, USA (1996)
5. Abrial, J.R., Butler, M., Hallerstede, S., Voisin, L.: An open extensible tool environment for Event-B. In: ICFEM. (2006) 588–605
6. Clements, P., Northrop, L., Northrop, L.M.: Software Product Lines : Practices and Patterns. Addison-Wesley Professional (August 2001)
7. Lee, K., Kang, K.C., Lee, J.: Concepts and guidelines of feature modeling for product line software engineering. In: ICSR-7, UK, Springer-Verlag (2002) 62–77
8. Lindner, T.: Task description. In Lewerentz, C., Lindner, T., eds.: Formal Development of Reactive Systems. Volume 891 of LNCS., Springer (1995)
9. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: Eclipse Modeling Framework. 2nd edn. The Eclipse Series. Addison-Wesley Professional (December 2008)
10. Antkiewicz, M., Czarnecki, K.: Featureplugin: Feature modeling plug-in for Eclipse. In: Eclipse '04: Proceedings of the 2004 OOPSLA, NY, USA, ACM (2004) 67–72
11. Poppleton, M., Fischer, B., Franklin, C., Gondal, A., Snook, C., Sorge, J.: Towards reuse with "Feature-oriented Event-B", Nashville, TN, In McGPLE (October 2008)
12. Bosch, J.: Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach. ACM Press/Addison-Wesley, NY, USA (2000)
13. Poppleton, M.: Towards feature-oriented specification and development with Event-B. In: Proc. REFSQ. LNCS, Trondheim, Norway, Springer (2007) 367–381
14. Cechticky, V., Pasetti, A., Rohlik, O., Schaufelberger, W.: Xml-based feature modelling. (2004) 101–114