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.

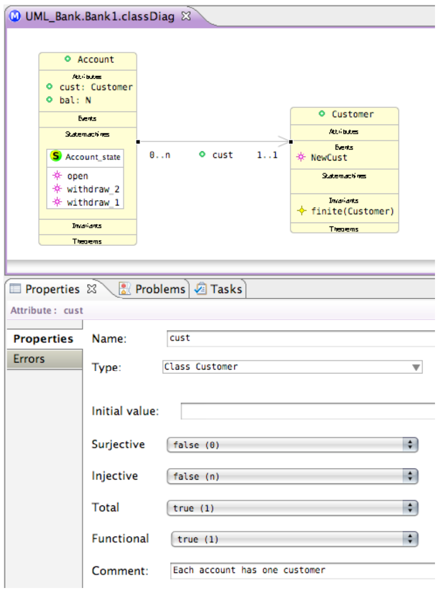
UML-B mirrors the Event-B approach where static data (sets and constants) are modelled in a separate package called a *'context'*. The Context diagram defines the static (constant) part of a model. The context diagram is similar to a class diagram but has only constant data represented by *ClassTypes*, *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)

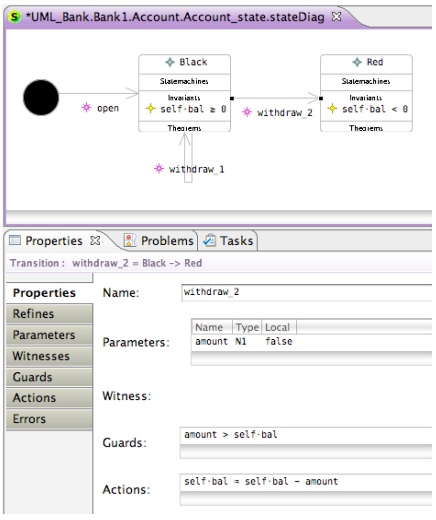
Injective: false (n)

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