UC-B: Use case modelling with Event-B

UC-B: Use Case Modelling with Event-B

Rajiv Murali, Andrew Ireland, and Gudmund Grov

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh, UK
{rm339,a.ireland,g.grov}@hw.ac.uk

Abstract. Use cases are a popular but informal technique used to define and analyse system behaviour. We introduce UC-B a plug-in for the Rodin platform (Event-B tool) that supports the authoring and management of use case specifications with both informal and formal components. The formal component is based on Event-B's mathematical language. Once the behaviour of the use case is specified, UC-B automatically generates a corresponding Event-B model. The resulting model is then amenable to the Rodin verification tools that enable system level properties to be verified. By underpinning informal use case modelling with Event-B we are able to provide greater precision and formal assurance during the early stages of design.

Keywords: Event-B, Rodin, UML, Use cases.

1 Introduction

UML use cases [1] are a popular but informal method for capturing behavioural requirements for software systems. They often appear in two complementary forms: (1) a *use case diagram* that provides an easy-to-understand illustration of the *subject*, *actors* and individual *use cases*, and (2) an informal textual *use case specification* that outlines a *contract* and *scenarios* for each use case. Errors introduced during use case modelling may later manifest themselves in the design or implementation phases where the cost of fixing them is significantly more expensive.

As shown in Figure 1, UC-B[1] is a plug-in for the Rodin platform [2] that supports a more formal approach to use case modelling. The tool allows users to provide formal counterparts to the informal use case specifications, i.e. pre- and post-conditions along with triggers and actions associated with the various flows (scenarios). The generic structure of the use case is then used to automatically generate an Event-B [3] model reflecting the natural levels of abstractions in the use case specification [4]. The generated Event-B model is then amendable to the verification tools provided by the Rodin platform. Furthermore, UC-B also supports the notion of an *accident case* which provides a way of representing potential accident scenarios and enables safety concerns to be explicitly taken into consideration during use case modelling [4]. An accident case is a sequence of

[1] Tool information on UC-B: https://sites.google.com/site/rajivmkp/uc-b.

actions that a system and/or other actors can perform that results in an accident or loss to some stakeholder if the sequence is allowed to complete. An important role of an accident case is communication – it allows designers to explain how the system behaviour will deal with hazards identified by safety engineers.

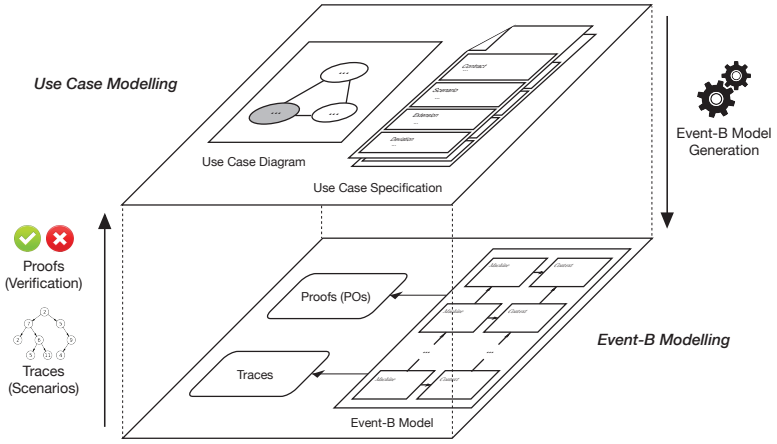

Fig. 1: The Rodin platform with UC-B.

This paper is structured as follows: Section 2 describes the process of using UC-B. Section 3 describes the tool architecture and future work, while our conclusion is presented in Section 4.

2 Using UC-B

Consider Figure 2 that presents a use case diagram for a *water tank system*. For simplicity we have included a single regular use case, i.e. MaintainH. UC-B provides its own data model that supports the creation of a use case model (UC-B model) on the Rodin platform. This UC-B model is allowed to contain use cases, actors and a subject that corresponds to those in the use case diagram (see Figure 3). The actors and subject are represented as *agents* in UC-B as described in Section 2.1. Each use case and agent are allowed to have name, label and informal description. The labels are required to be unique as they are later used to help provide traceability between the use cases and its generated Event-B model.

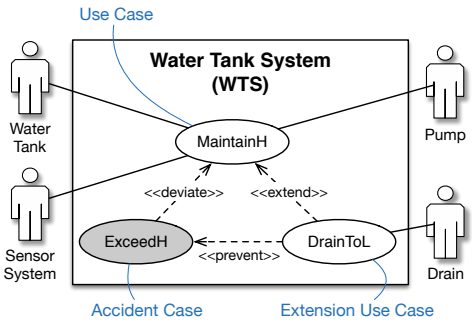

Fig. 2: Use case diagram.

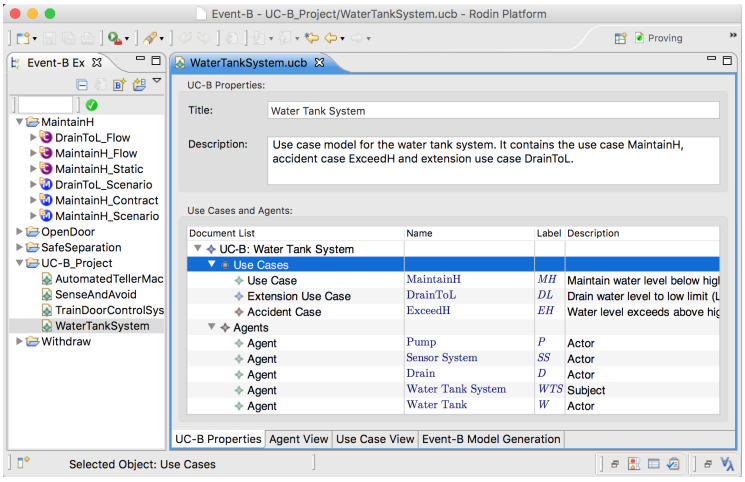

Fig. 3: UC-B model on Rodin: Use cases, actors and subject.

2.1 Agents

Each agent is allowed to specify a distinct list of *variables*, *constants* and *carrier sets* that are associated to the domain of the actor or subject it represents. These provide the basis for the formal aspect of the use case specification.

In Figure 4, the agent Water Tank introduces the variable *waterlevel* that represents the dynamic water level in the tank. Each variable specifies a predicate and assignment that denotes its type and initialisation respectively. For example, the water level has the type $waterlevel \in L..H$ and initialisation $waterlevel := H$. The constants L, LT, HT and H denote the low limit, low threshold, high threshold and high limit of the water tank. Each constant specifies a predicate that denotes its type, e.g. the type for L is $L = 0$ and LT is $LT > L$. UC-B relies on the Rodin platform to check for well-formedness of predicates and assignments.

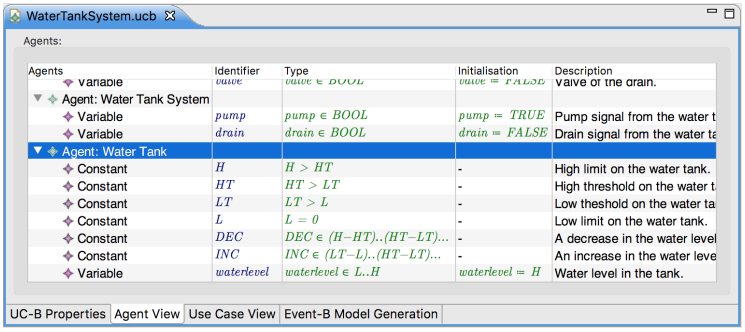

Fig. 4: Agents with constants and variables.

2.2 Use Cases

Each use case contains a contract and scenarios as seen in Figure 5. The contract allows the user to specify constraints that apply to the execution of the use case, i.e. pre-conditions, post-conditions and invariants. For example, the pre-condition and post-condition for use case MaintainH is formally specified as $waterlevel > HT \wedge waterlevel \leq H$ and $waterlevel \geq L \wedge waterlevel \leq HT$ respectively. These provide constraints on the variable *waterlevel* that are required to be satisfied before and after the execution of the use case MaintainH.

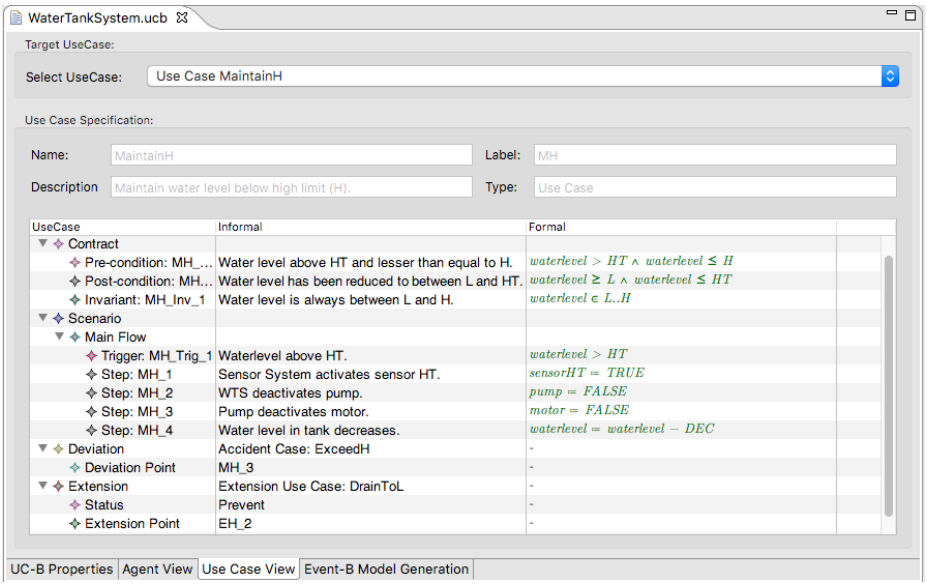

Fig. 5: Use case specification of MaintainH with contract and scenario.

The scenario is represented by a trigger and a sequence of steps (flow). The trigger describes the condition that can initiate the execution of the flow. A step may represent an action that allows the user to specify an assignment that modifies a variable. For example, the final step MH_4 in the scenario of MaintainH reduces the water level via the assignment $waterlevel := waterlevel - DEC$. The behaviour introduced in the scenario is required to satisfy the contract of the use case.

The tool also supports the use of accident cases and extension use cases. The scenario of an accident case is specified as a *deviation* from a regular use case. The user is allowed to specify a step in flow of a use case where the deviation to the accident scenario may occur. For example, the scenario of the accident case ExceedH is specified as a deviation at step MH_2. If the scenario of the accident case is allowed to complete the system results in an accident, i.e. the contract of the use case is expected to be violated.

In use case modelling, an extension use case is often used to describe how a system should respond in exceptional circumstances. The behaviour of an extension use case can be introduced between the steps of a scenario via an *extension-point*. UC-B allows the user to specify a step before which the behaviour of the extension use case can be introduced. In the example, the extension use case DrainToL is introduced before the final step of the accident scenario EH_2. A full description of this example is given in [4].

2.3 Event-B

As noted above, once the use cases have been specified, UC-B supports the automatic generation of a corresponding Event-B model. Figure 6 illustrates the structure of the Event-B model generated for MaintainH. All static aspects in the use case model are introduced in the context MaintainH_Static. This context is *seen* by all machines generated for the use case.

The contract associated with the MaintainH use case is represented in the initial Event-B machine, i.e. MaintainH_Contract. The initial machine is refined by MaintainH_Scenario which represents the main scenario associated with MaintainH. Since the refinement is formally verified, the MaintainH scenario is guaranteed to satisfy the given contract. Any deviations and extension-points in the use case scenario result in accident and extension use cases to be taken into account in the Event-B model respectively. The accident scenario for the accident case ExceedH is introduced in the machine MaintainH_Scenario as a deviation from the ideal scenario. Extension use cases result in a further refinement, e.g. DrainToL_Scenario. The refinement style of modelling promoted by Event-B reduces the complexity of proofs and thus increases proof automation.

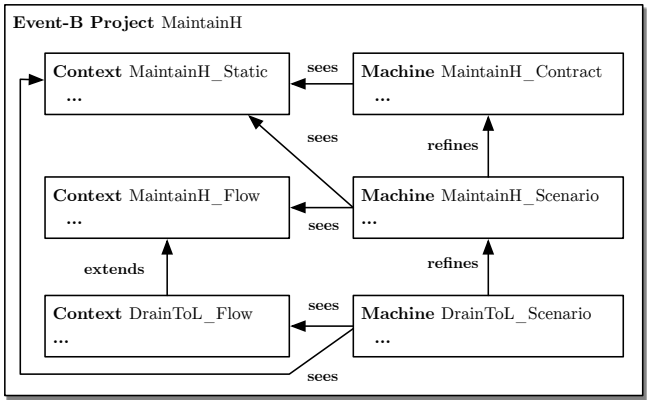

Fig. 6: Structure of Event-B model for MaintainH.

3 Architecture and Further Developments

Figure 7 shows the relationship between UC-B and the Rodin platform. The UC-B meta-model is structured using the Eclipse Modelling Framework [5]. APIs provided by Rodin enable UC-B to detail the content of the UC-B model using Event-B's mathematical language and support the subsequent generation of Event-B models. Further development is being undertaken to integrate UC-B with the Papyrus [6] Eclipse-based tool that supports standard UML2 modelling environment. The integration is aimed at relating use case specification detailed in UC-B with other downstream UML diagrams.

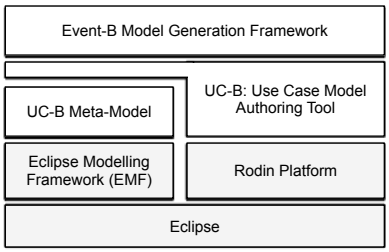

Fig. 7: UC-B architecture.

4 Conclusion

Building upon the Rodin platform for Event-B, we have described the UC-B tool which supports the authoring and management of use case specifications. UC-B focuses in particular on the textual aspects of use cases. Specifically users are required to specify a formal counterpart to the informal text that is normally provided. The pay back comes through the automatic generation and verification of a corresponding Event-B model. UC-B also introduces the notion of an accident case which provides a mechanism for explicitly representing and reasoning about potential accidents during use case modelling.

Acknowledgements: The first author was supported by an Industrial CASE studentship which was funded by EPSRC and BAE Systems (EP/J501992), while the second and third authors were partially supported by EPSRC grants EP/J001058 and EP/N014758/1. We also would like to thank Benjamin Gorry, Rod Buchanan and Paul Marsland from BAE Systems.

References

1. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language. Addison-Wesley (1997)
2. Abrial, J.R., Butler, M., Hallerstede, S., Hoang, T.S., Mehta, F., Voisin, L.: Rodin: An Open Toolset for Modelling and Reasoning in Event-B. International journal on software tools for technology transfer **12**(6) (2010) 447–466
3. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press (2010)
4. Murali, R., Ireland, A., Grov, G.: A Rigorous Approach to Combining Use Case Modelling and Accident Scenarios. In: NASA Formal Methods. Springer (2015) 263–278
5. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: eclipse modeling framework. Pearson Education (2008)
6. Gérard, S., et al.: Papyrus uml. URL: http://www. papyrusuml. org **8** (2012)