A Framework for Diagrammatic Modelling Extensions in Rodin

Vitaly Savicks and Colin Snook

University of Southampton

The Event-B language is purposely designed to have minimal features to support automatic proof. In this sense the language can be seen as low-level in some problem domains. It is attractive therefore to enhance the language with higher-level modelling constructs (often using diagrammatic notations) and automatically generate Event-B for verification. This was the motivation behind the State machines plug-in. The Eclipse Modelling Framework (EMF) provides supporting infrastructure for building modelling tools. An extensible EMF framework for the Event-B language is already available. Further frameworks are now available to support language extensions and diagrammatic editors.

These frameworks have been developed to support higher-level diagrammatic models which contribute to an Event-B model. For verification purposes, model extensions are converted into pure Event-B. This is done within EMF, prior to persisting into the Rodin database. An alternative approach would be perform the translation in Rodin, or even to extend the verification tools to work directly upon the higher-level modelling constructs. However, the disadvantage of these approaches is that the model extensions have to be replicated within the Rodin DB. By translating to Event-B within EMF we avoid the need for Rodin to comprehend the higher-level model and persist it as a single string attribute of a generic Rodin element defined for this purpose. A generic facility provides load/save operations of the higher-level model extension. All that is required to invoke this facility is a declaration of the meta-class of the root element of the model extension.

Diagram extension to Event-B meta-model introduces two important abstract classes – Diagram and DiagramOwner – to be used for any diagram tool built on top of Event-B. These are high-level abstractions for any particular diagram meta-model that a developer decides to implement for his specific tool. Diagram class is an abstraction of a diagram element that would usually map to a canvas in a concrete diagram editor. DiagramOwner, in turn, is an element of a diagram that can contain other diagrams, as it has diagrams containment attribute. These two simple abstractions enable any concrete diagram to have a common type and to contain other diagrams that use this same extension.

The Rodin tool provides an Event-B navigator view that shows the contents of Event-B models in a tree structure. The problem with this view that an EMF developer of Event-B extension would face is that it only displays Rodin elements and doesn't know how to treat EMF elements. This problem is generic, as is generic a solution we've provided. The developer is merely asked to provide an adapter factory extension for his specific domain meta-model. The implementation of the required adapter factory is available from generated EMF.Edit

plug-in. Thus, navigator view will be aware of new extensions to Event-B and how to display them with appropriate labels and icons.

As long as a diagram extension is an extension of Event-B EMF, the navigator support mentioned above works for free, provided a developer has specified a correct adapter factory. However, diagrams are more than just extensions to Event-B when used in a diagram editor, combining both domain model and graphical notation, which are kept in separate resources. The navigator support for diagrams provides an extension point to diagram editor developers with an interface of a diagram provider that, when implemented, enables opening a diagram from navigator on double click. Potentially the same provider can offer additional functionality for a diagram from navigator view.

Refiner - The Rodin platform provides a utility to automatically make a refinement from an Event-B machine as a starting point for modification. It is necessary to provide this facility for language extensions. The extensions framework extends the Rodin refine utility with a generic utility that reflectively performs a deep clone of a given part of the model. This is invoked for a particular modelling extension by a declaration in the plugin. Although cloning can be handled reflectively, references vary in the way they are handled and therefore the plugin also needs to contribute a class which defines how to handle references.

Generator - A generic Event-B generator is provided to simplify the task of defining a transformation from a model extension to Event-B. The generator is declared giving the meta-class of the root source element of the modelling extension. Rule classes are defined and declared for each meta-class that the generator is intended to work for. Each Rule must define 3 methods:

1. enabled - returns a boolean to indicate whether the rule is appropriate for a given source element
2. dependenciesOK - returns a boolean to indicate whether any pre-requisite generation has been completed by other rules. (If not the rule will be deferred and tried again later).
3. fire - returns a list of definitions which describe how references and containments need to be altered as part of the generation.