Code Generation – Tool Developments:

Andy Edmunds
University of Southampton
ae2@ecs.soton.ac.uk

[1] Industrial Experience.

We collaborated on an industry-led assessment of Rodin/Event-B, and Tasking Event-B for code generation, with Thales Deutschland. The feasibility study involved modelling controller software, and the operating environment, of a simple fan controller. The aim of the study was to evaluate the complete Event-B methodology, from abstract specification to Java implementation. During the project various tool enhancements were completed. A list of 'significant' (Code Generation) feature requests was submitted, from suggestions made by Thales' engineers. They consider these features necessary for the tool to be useful in a production environment. The list could be used to guide future research and development.

[2] Java Translation Improvements.

The tool was updated (Sept 2013) largely driven by the activities of [1]. Improvements were made to the translators for generating Java code, including bug-fixes, and enhancements aimed at improving usability. Such as setting up the project with a Java Nature, and generating code in the current project. We added automatic flattening of invariants, and events; and automatic inference of typing, and parameter direction annotations. We also added environment interfaces, which simplifies the process of specifying interactions with driver software in the environment. The enhancements mean that a developer has to perform fewer steps, to generate code from an appropriately constructed model. The generated code can often be run in the same project as a Java application, with very little further configuration.

[3] C Translation: for Co-simulation with FMI.

The Functional Mock-up Interface (FMI) is a framework supporting co-simulation of distinct Functional Mock-up Units (FMUs). Recent work allows Event- B models to be co-simulated with FMUs, using ProB2. Existing code generation for Event-B supports generation of embedded controller implementations. We adapted the existing Event-B code generators, to produce code for use in FMUs. Controller code, generated from Tasking Event-B, can be compiled and packaged in an FMU. This allows generated controller code to be tested with a continuous model of the environment (in ProB2). Alternatively, it can be imported into a 3rd party simulator.

[4] Templates for Configuration and re-use.

This work arose from a feature request in [1]. The template-driven approach is a partial solution for tailoring code to be deployed on a specific target platform. The templates contain ‘boilerplate’ code, which would otherwise need to be hard-coded in the translator. We developed a tool to merge the code templates with code generated from the formal model.

We developed a lightweight approach, where tags (i.e. tagged mark-up) can be placed in the source templates. A 'generator' interface can be implemented to generate code, or additional information, when a tag is encountered. The template-processors may be of use to other plug-in developers, when wishing to merge an annotated text file with some automatically generated output.

[5] Theories for implementable Sets and Functions.

Code generation involving Event-B sets and functions has largely been avoided, in our work, until now. We have recently been looking at this issue, and have created Theories for implementable sets and functions, using polymorphic type parameters to describe implementations involving generic (parametrised) sets and maps.

For sets, we introduce operators for typing, and initialisation. We support a number of set operations (such as union/subtraction/intersection) and introduce new operators to facilitate code generation. A translation, to Java, of the new *implementable* type and operators is also defined in the theory. This is supported by a Java implementation of a set, which is intended to be a refinement of the model.

Functions are similar in that we introduce operators for typing, and initialisation, and a Java translation. We also add update and lookup operators. The translation to Java is backed by an implementation using a HashMap to store the domain and range values, as key-value pairs.