Developments in Code Generation Tools for Event-B

Andy Edmunds

April 2, 2013

Since the last Rodin Workshop we have been working on a number of aspects of Code Generation. We will give an overview of the work that we have undertaken, and what we are doing currently.

1 State-machine translation

Since the last workshop we added the ability to generate code from Event-B state-machine diagrams; we will give an overview of the approach. Tasks in Tasking Event-B can be used to generate code for embedded controllers. The simulation approaches makes use of a single task, which simulates concurrently executing state-machines. We are then able to instrument the code to guide the simulation, to improve coverage analysis on the controller code. Code generation for state-machine generates the following constructs:

- A case construct, for each state-machine.
- A case statement in the construct, for each state in a state-machine.
- A branch for each transition from a state.
- A branch condition that can be used to guide the simulation.

The main program invokes the state-machine implementations in a loop, once per cycle. Each iUML-B state-machine diagram maps to a procedure. State-machine procedures are called exactly once before the sends to, and reads, from the variable store. The evaluation of each state-machine procedure is independent of every other state-machine, since each state-machine keeps a local copy of the state, copied from the variable store. Each state-machine procedure has a n state variables v, representing states s_i in the state-machine diagram, where $i \in 1 \ldots n$. During code generation we create a procedure for each state-machine, Each procedure has a case statement (with pseudo-code statement A_i), which has the following form:

$$
\begin{array}{l}
\textbf{procedure } \mathit{statemachine}1(\ )\{ \\
\quad \textbf{case } v = \ s_1 \textbf{ then } A_1; \\
\qquad\qquad\ s_2 \textbf{ then } A_2; \ldots \\
\qquad\qquad\ s_n \textbf{ then skip}; \\
\quad \textbf{end case } \}
\end{array}
$$

In the current code generation tool, each of the n outgoing transitions of a state, is elaborated by an event $Event_i$. The program statement arising from $Event_i$ is a branch in A_i, with a condition g_i, and an update a_i. The following shows the branching style, where skip leaves the state unchanged:

$$
\begin{array}{l}
\textbf{case } v \\
\quad s_1 \textbf{ then} \\
\quad\quad \textbf{if } g_1 \textbf{ then } a_1; // \ from\ transition\ 1\ (Event_1) \\
\quad\quad \textbf{elseif } g_2 \textbf{ then } a_2;\ // \ from\ transition\ 2\ (Event_2) \\
\quad\quad \textbf{elseif } g_i \textbf{ then } a_i;\ // \ from\ transition\ i\ (Event_i) \\
\quad\quad \textbf{else}\ \ \textbf{skip} \ldots \\
\textbf{end}\ \ \textbf{case}
\end{array}
$$

By adding further guards to g_i we are able to guide the simulation to improve coverage.

2 FMU translation in C

The ADVANCE project aims to simulate cyber-physical systems modelled using Event-B. The Functional Mock-up Interface (FMI) approach is being used to support co-simulation of dynamic models, using a combination of XML-files and compiled C-code. The system being simulated can consist of a number of Functional Mock-up Units (FMUs), all of which are under the control of a master simulator. All communications between the FMUs takes place via the master. The master algorithm works cyclically; FMU data is read by the master, and distributed to FMUs. Each FMU is instructed to perform a processing step by the master. The cycle repeats for a specified time period after which the simulation is complete.

From a code generating viewpoint we are interested in generating controller FMUs from Event-B models. The environment may then be simulated in ProB, or another FMU (including environment code that we generate). The output generated for the controller FMU is contained in a zip file with the following contents:

- an XML model description file,
- C code, and compiled libraries.

We will explain briefly how this can be achieved.

3 Other Work

We are currently involved in a project with Thales Transportation Systems GmbH, in Germany. The project is in its early stages, and we are hoping to make use of the code generation capabilities of Tasking Event-B. We will report on progress and the issues arising from this work.