Statechart Code Generator Releases

(Last Updated: September 20, 2012,
Latest Accurev Snapshot:
INTERNAL_JPL_Release_5Aug2011)

This is an internal JPL web page providing the latest releases of the JPL Statechart Autocoder. Release of the Python only tar file can be found at the Python Statechart Code Generator page.

Downloads:

Actually the ***autocoder.tar*** code generator tool included in the tar files is capable of generating Promela (design verification language), C/C++ state machines based on the Quantum Framework public domain infrastructure, in addition to Python. We currently do not have a user's manuals for ether the Promela or C/C++ code generation but it follows roughly the same process as for Python. The following tar files are bundles with support for languages:

- 5 August 2011 Python/Promela Statechart Autocoder Release Downloads
- 5 August 2011 Python/C++/C and Promela Statechart Autocoder Release Downloads

NOTE: Currently the –m32 flag in the C++ Makefile generated is missing from the link line so that C++ does not build properly on 64 bit machines. One can easily add this flag to fix this problem.

Release Notes:

1. The JPL Statechart Autocoder should work on Mac OS X, Red Hat Linux and Windows XP. Some uses use Cygwin to run it under Windows but while this can work we are not currently

recommending it. For each of the above releases we recommend verification that Autocoder.jar works properly to generated the desired language properly by running unit tests that come with each language support project. To run the unit tests perform the following steps:

1. Execute the following to change directory to QF_Py/bin

   cd <root>/QF_Py/bin

   Set your $QFROOT environmental variable to point to the <root>/QF_Py path. Then source ether cshrc.sh or bashrc.sh to set up the PYTHONPATH environment variable for executing the unit tests.
2. The unit tests for each language can be executed separately as follows:

   For Python: <root>/QF_Py/bin/pythonsuite.py
   For Promela: <root>/QF_Py/bin/spinsuite.py (Note that Spin must be installed on your machine)
   For C++: <root>/QF_Py/bin/cppsuite.py
   For C: <root>/QF_Py/bin/csuite.py

   If you are testing over all the languages you can also use <root>/QF_Py/bin/allsuites.py. You will see various messages displayed but at the end of the tests you should see a summary message displayed that shows all tests have PASSED.
3. Some of the test sets are a bit long so if you desire it is possible to execute a select subset of tests. IT IS RECOMMENDED ON NEW INSTALLS THAT YOU RUN ALL TESTS FOR YOUR TARGET LANGUAGE ATLEAST ONCE TO VALIDATE OPERATION. To display a help message and list of tests to execute you can use the "-h" option with any of the scripts listed in 2. For example execute "spinsuite.py –h" will display at the bottom of the help text a list of tests like this:

   Available tests: SpinSuite1 ['testSimple1', 'testSimple2', 'testSimple3','testSimple4'] SpinSuite2 ['testComposite01',

'testComposite02','testComposite03', 'testComposite04', 'testComposite05a', 'testComposite05b','testComposite06']

You can execute for example "spinsuite.py SpinSuite1.testSimple3 SpinSuite2.testComposite02". This will result in lots of test stuff being displayed but at the end should look like this:

Final Test Results:

======== Spin Simple Test Suite ========
Simple3, Junction, Junction Transition: PASSED

======== Spin Composite Test Suite ========
Composite2, Hsm2, Deeper composite state and transitions: PASSED

Any of the test scripts can be used this way.

Promela Features Supported:

The following UML State Machine Features are supported in the Promela within this release:

- Simple States (or Leaf States)
- Composite States
- Orthogonal Regions in States
- Initial and Final Pseudo-States
- Junction and Choice Pseudo-states
- Transition
- Transition to self
- Guards (using stubbed, user-populated Promela inline functions)
- Actions (Entry, Exit, Transition, and signalEvent, using stubbed, user-populated Promela inline functions)
- Timer Events

Note that each of these features has a unit test that validates the generate Promela code. A working version of Spin for your platform is required to

run the Promela but no Quantum Framework is needed.

Fixes added to Promela 30 July 2011:

Promela bug fixes for internal transition handling and missing entry/exit actions:

1. Fixed PromelaStateMachineWriter::makeState to consider an event signal processed only when it appears on a transition whose source state is the current state being processed
2. Fixed PromelaStateMachineWriter::makeStateTransition to more correctly handle juntion/choice branches with multiple outgoing transitions: each guard condition needs its own else branch in Promela
3. Fixed FlattenedVelocityModel to exclude exit action of source parent state in state hierarchy _if_ source state is a pseudostate; likewise, include entry action of target parent state _if_ target state is a pseudostate
4. Added an addUnique operation to AbstractVisitor to avoid adding the same element multiple times to a visitor list: VertexVisitor and LeafStateVisitor fixed to not produce duplicate states in Promela
5. New UMLValidation rule added to abort for the case where a composite state has no initial state within, but there is at least one incoming transition to the composite state

C++ Features Supported:

The C++ generation produces state machine code compatible with Quantum Framework 4.x that is included with the tar file in the Cpp project. Pretty much all the Python features are supported except for Threaded do-Activity was intentionally left out to make the code comply with JPL flight practices. It is important to note that this version of the tool does not produce JPL flight code but rather each flight project produces it's own backend to the tool that generates their specific style of C code.

The software was developed for the Ares1 Model Validation Task and SMAP Flight Software development. The Python generation has been successfully used on Ares1 is used on SMAP to support Fault Protection

System engineers doing function requirements level simulation. At this time the Python code generation is no longer being actively developed, however, if a bug is discovered we are very interested to fix it and have the software used by others. We invite you to evaluate the tool for your project. We are interested to hear your feedback and suggestions for other tools we might develop in the future.

If you have questions about the tools, suggestions for feature additions or wish to report a problem send email to the following people:

Garth Watney
watney@jpl.nasa.gov

Leonard Reder
reder@jpl.nasa.gov

Shang-Wen (Owen) Cheng
Shang-wen.cheng@jpl.nasa.gov