The International Conference on Dependable Systems and Networks
San Francisco, CA, June 22nd - 25th, 2003
Workshop on Model-Checking for Dependable Software-Intensive Systems
Abstract Submittal

Title: Experiences in Integrating Auto-Translated State-Chart Designs for Model-Checking

Authors/Participants: Paula J. Pingree (Paula.J.Pingree@jpl.nasa.gov)
Edward G. Benowitz (Edward.G.Benowitz@jpl.nasa.gov)
Autonomy & Control Section
Jet Propulsion Laboratory
Pasadena, CA 91109

1.0 Background

In the complex environment of JPL's flight missions with increasing dependency on advanced software designs, traditional software validation methods of simulation and testing are being stretched to adequately cover the needs of software development. Our aim is to apply formal method techniques and tools to validate mission-specific components of flight software that are specified using finite state machine representation. We have established an automatic translation toolset called ***HiVy*** that translates Stateflow® state-charts to Promela, the input language of the Spin model-checker, for the validation of mission-specific components. To guarantee compliance with code that may be auto-generated from these state-charts our translation toolset preserves the Stateflow® semantics. We are now able to specify and validate portions of mission critical software design and implementation using the exhaustive exploration techniques of model-checking.

2.0 Closed-Loop Model Integration: How we do it and why?

Figure 1 presents our system that accepts state-chart specifications both in graphical and tabular notation for translation into Promela. Hierarchical Sequential Automata (HSA) is an intermediate format that offers a set of elements for defining the syntax and semantics of Stateflow® charts. The HiVy Toolset implements the HSA format to enable translation of Stateflow® model designs. A tool paper on the HiVy Toolset has been submitted

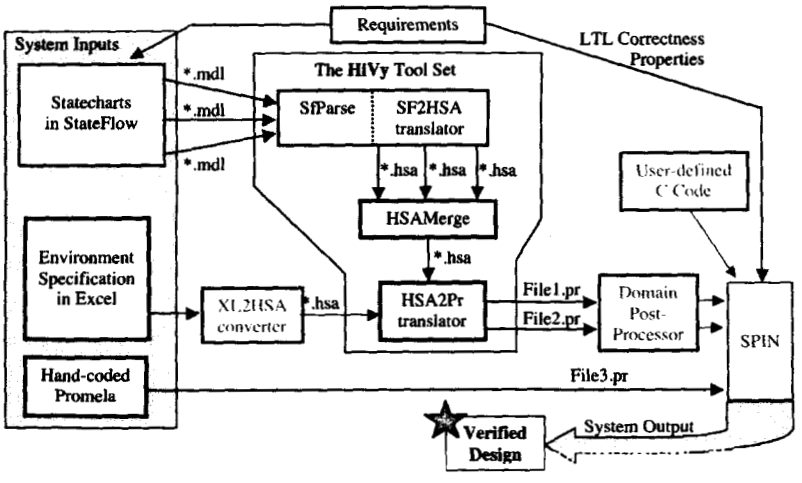

Figure 1. The HiVy Toolset System

for the 10th International Spin Workshop to be held in May, 2003 in Portland, OR. For this Workshop we focus on the integration components (in red) of our system that we found necessary to close-the-loop around our HiVy-translated designs in order to provide an acceptable model to Spin for validation.

2.1 Xl2Hsa

While specifying a graphical state-chart of a design is convenient for the system engineer, a simpler approach is possible if one is only interested in creating the environment model for verification. In most cases, the environment model will only involve states, hierarchies, and transitions. This is an adequate subset of Stateflow®. We have developed a tool called Xl2Hsa that allows the expression of such state machines in a tabular format. Excel is used to easily produce and edit such specifications. The user is required to describe states in terms of their hierarchy and parallelism. The user then describes transitions by specifying the starting state and destination state of that transition. By using this tabular format, state models can be created without the more intensive process of drawing transitions and states in a graphical manner.

2.2 Post-processor

By default, HiVy generates Promela code to model the state machines with Stateflow semantics. However, certain applications may wish to modify the semantics to incorporate application or domain-specific behavior. In the case of DS1, specific scheduling behaviors were added, which were indicated by specific function calls to C code. A domain-specific post-processor was created in Perl. Using regular expressions, scheduling calls to the C-code were replaced with calls to Promela code that modeled the domain-specific scheduling semantics. An additional use of the postprocessor can be to add non-determinism in specific cases. For example, a transition waiting for a specific timeout to occur can instead be replaced with a "true" transition in Promela. A simple find-and-replace postprocessor can be used in this situation.

2.3 C-code and the Environment Model

We rely on the availability of a newly extended version of the Spin model-checker (Spin version 4) that allows for the use of embedded C code fragments inside Promela code. Via this mechanism, the model-checking code can be linked with original C code libraries that implement elements of the flight software that can be executed as atomic functions during the model-checking exercise. However, C code is in general not visible by the model-checker, so we minimize the amount of C code used in our integrated system. Our process is to first stub out the necessary C code, to find the minimum level of C code needed to allow the full Promela model to be compiled. We then add small amounts of C code to manipulate the Promela environment model, using the *now* variable. In practice, several iterations are needed to ensure that the environment model has enough states to model the interactions presented by the C code. For our case, the environment model typically consists of small state machines having 2-4 states, but this will vary depending on the system's complexity.

2.4 Non-determinism

Even after pre-processing, a certain amount of Promela hand coding may be necessary, due to the need to add non-determinism. For example, when in a particular state-chart state, transitions out of the state may depend on either a timeout signal or a completion signal. In Promela, we want to test all possibilities therefore we need to non-deterministically set these signals. In such a case, hand coding may be necessary to add the non-determinism if a postprocessor is not suitable for the task.

3.0 Conclusion

The integration lessons learned in completing our closed-loop Promela model have specific ties to our domain design and automated translation methods. However the challenge to integrate a closed-loop system for Spin model-checking is a widespread issue. We hope to share our experiences and learn more about this and related topics in model-checking through participation in the *Workshop on Model-Checking for Dependable Software-Intensive Systems.*

Acknowledgment

The research described in this abstract was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.