A Flexible Statechart-to-Model-Checker Translator

[System Demonstration Proposal for ISRE'99]

Nicolas Rouquette, Julia Dunphy & Martin S. Feather
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA
+1 818 354 1194
Nicolas.Rouquette@Jpl.Nasa.Gov, Julia.Dunphy@Jpl.Nasa.Gov & Martin.S.Feather@Jpl.Nasa.Gov

OVERVIEW

Many current-day software design tools offer some variant of statechart notation for system specification. We, like others, have built an automatic translator from (a subset of) statecharts to a model checker, for use to validate behavioral requirements. Our translator is designed to be *flexible*. This allows us to quickly adjust the translator to variants of statechart semantics, including problem-specific notational conventions that designers employ.

Our system demonstration will be of interest to the following two communities:

- **Potential end-users:** Our demonstration will show translation from statecharts created in a commercial UML tool (Rational Rose) to Promela, the input language of Holzmann's model checker SPIN. The translation is accomplished automatically. To accommodate the major variants of statechart semantics, our tool offers user-selectable choices among semantic alternatives. Options for customized semantic variants are also made available. The net result is an easy-to-use tool that operates on a wide range of statechart diagrams to automate the pathway to model-checking input.
- **Other researchers:** Our translator embodies, in one tool, ideas and approaches drawn from several sources. Solutions to the major challenges of statechart-to-model-checker translation (e.g., determining which transition(s) will fire, handling of concurrent activies) are reified in a uniform, fully mechanized, setting. The way in which the underlying architecture of the translator itself facilitates flexible and customizable translation will also be evident.

MOTIVATION FOR TRANSLATOR

We see increasing use of statechart-like notations, in our environment for specifying portions of spacecraft. These notations are both precise and user-friendly. Precise means they can be manipulated mechanically (e.g., by automatic code generators). User-friendly means they are readily adopted by a wide variety of users, an important asset in development of spacecraft, since experts from diverse areas (e.g., navigation, power, telecommunications) are involved.

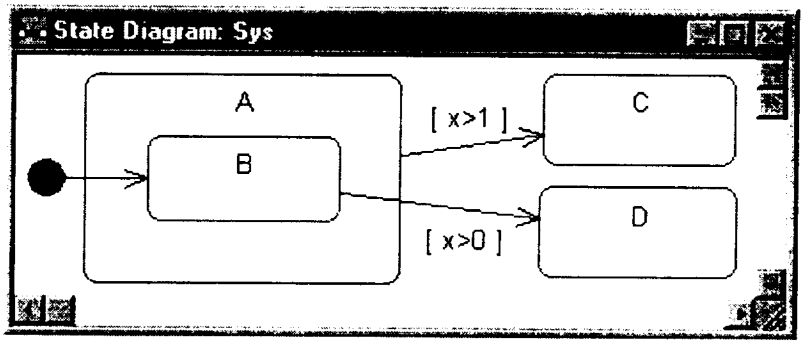
The critical nature of spacecraft control software warrants thorough validation and verification to assure its correct operation. Its state-based nature offers admits analysis via state-exploration (a.k.a. "model checking") techniques. Indeed, our colleagues have demonstrated the practical utility of this analysis technique applied to spacecraft software [Schneider et al, 1998], [Lowry et al, 1997].

Our aim is to make such make model-checking a routine and inexpensive analysis activity. We have taken our inspiration from several efforts within the research community that address the automatic translation from state-based notations to the form of input accepted by model checking tools. In particular, [Mikk et al, 1998] translate a substantial subset of Statemate statecharts to both SPIN and SMV. [Day, 1993], also addresses the model checking of STATEMATE statecharts. [Latella et al, 1999], who focus on model checking of UML statechart diagrams, similar but not identical in meaning to Harel's original statecharts [Harel, 1987]. We are also aware of the translation to model checking from other state-based notations, notably SCR [Heitmeyer et al, 1999], and RSML [Czerny & Heimdahl, 1998].

We are driven by the pragmatic concerns of analyzing actual spacecraft designs. We encounter statechart-like notations from a variety of sources, for example, statecharts created in MATLAB®, Rational Rose®, Statemate®, and Visio®[1]. The upsurge of interest in UML, and the fact that the semantics of UML's statecharts differ from the semantics of Harel's classical statecharts, means that there are now at least two prominent variations of statecharts in current usage. For complex cases of either notation, the semantics can be quite subtle (e.g., see [Harel & Naamad, 1996] for an article whose purpose was to clarify STATEMATE's semantics for statecharts). Furthermore, we have seen design groups inventing their own notational conventions, different from both Harel and UML statecharts! In response to these concerns, we have constructed our statechart-to-model-checker translator to be *flexible*, so that we can rapidly adjusted it to variants of statechart semantics as needed. Our tool is seamlessly integrated with a commercial UML design tool. Through this combination of features we realize a useful and usable system.

BRIEF TRANSLATION EXAMPLE

The figure shows a simple example of multiple enabled transitions emerging from nested states. If state machine Sys is in state B, and condition x>1 is true, then two transitions are enabled: the transition from B to D and the transition from A (a superstate of B) to C (since if x>1, then x>0). Statecharts define *priority* between such transitions:



- In classical statecharts, transitions from a superstate have priority over transitions from a substate of that superstate. Of all the enabled transitions, only those of the highest priority are taken. Thus, for the example, classical statecharts would select the transition from A to C.
- In UML, the priority is the other way around. That is, transitions from a substate have priority over transitions from a superstate of that substate. Thus, for the example, UML statecharts would select the transition from B to D.

The Promela code generated by our translator considers transitions in order of their priority, highest first. For classical statecharts semantics, this is as follows ("..." indicates code fragements omitted for brevity of presentation):

[1] MATLAB ® The MathWorks Inc, Rational Rose ® Rose Corporation, Statemate ® I-Logix Inc, Visio ® Visio Corporation.

```
...
if
:: IN_STATE_Sys_B ->
 if
   :: x>1 -> Sys_Response1 = T
   :: else ->
     if
       :: x>0 -> Sys_Response2 = T
       :: else -> skip
     fi
 fi
...
:: Sys_Response1 -> Sys_state = Sys_D ...
:: Sys_Response2 -> Sys_state = Sys_C ...
...
```

The transition from A to C is considered first – if enabled, it will cause Sys_Response1 to occur, namely the transition to state D. Using Promela's "if ... else ... fi" construct, we ensure that the transition from B is considered only if all the transitions from A have been determined to be un-enabled.

We also offer the option of a simple translation that imposes no prioritization among transitions from nested states. That is, it is possible to take any such enabled transitions.

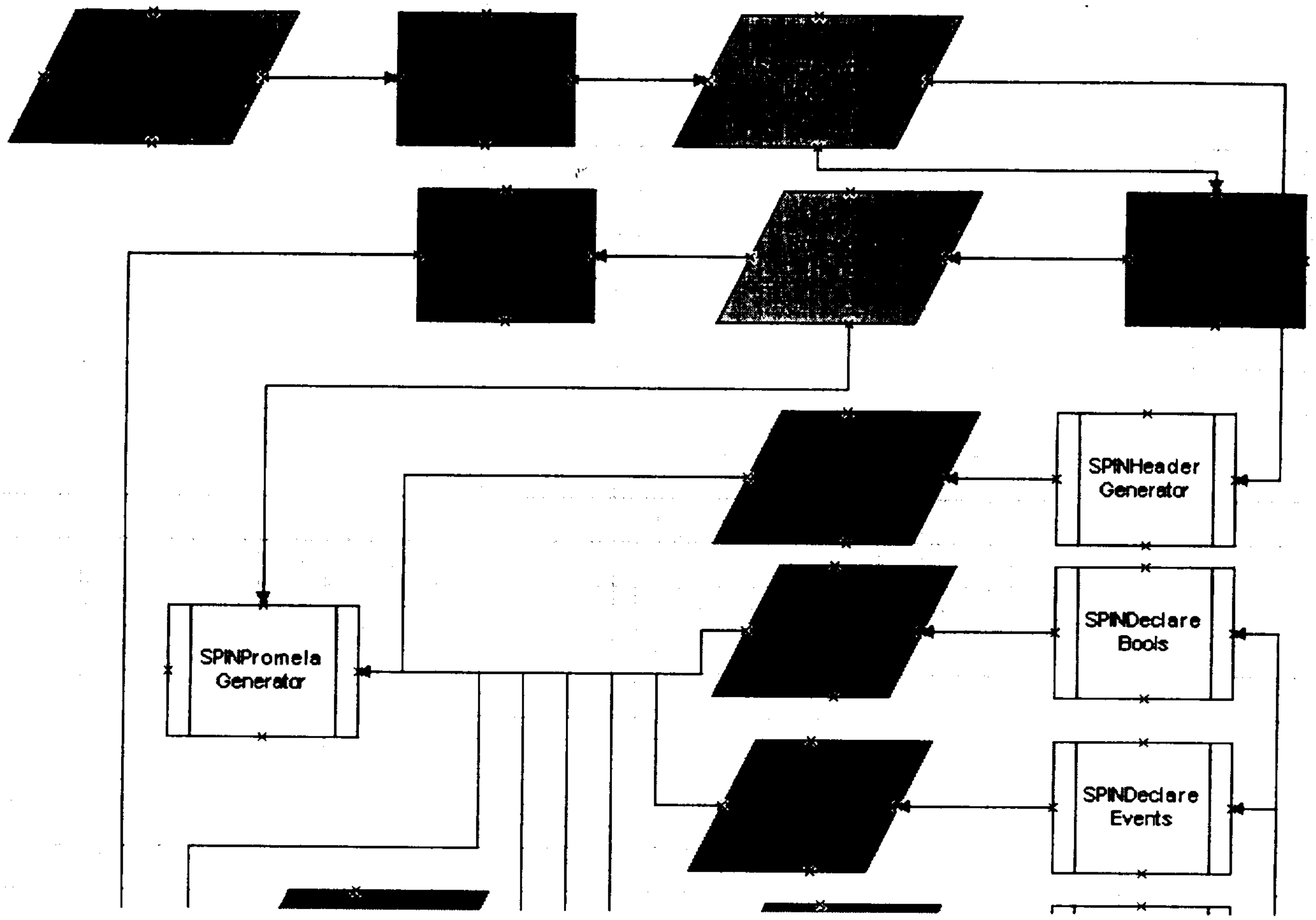
```
...
if
   :: x>1 ->  Sys_Response1 = T
   :: x>0  -> Sys_Response2 = T
   :: else -> skip
fi
...
```

This option can be used to check safety properties that should hold regardless of the priority scheme adopted. When SPIN is executed on such Promela code, all the possible transitions will be explored.

The above fragments also illustrate our treatment of statechart concurrency – in order that the enabledness of transitions of concurrent statecharts (or concurrent substates) be correctly computed, our translator separates the computation of which transitions are to take place from their subsequent execution. (In the Promela fragments above, the Sys_Response1 and Sys_Response2 variables are booleans serve to communicate between these two phases).

ORGANIZATION OF THE TRANSLATOR FOR FLEXIBILITY

Briefly, our translator is organized as a network of information processing nodes and information repositories. The figure on the next page shows a fragment of the translator network. In our current implementation we use Visio as the front-end GUI, which connects to Visual Basic to execute the translator itself. The network orchestrates the translation. At the back end, Visual Basic connects to the design tool (currently we are using Rational Rose) to access statechart information. This organization facilitates an expert user in re-arranging the translation on a case-by-case basis in response to semantic needs.



CONCLUSIONS

Our translator is an implemented system, integrated with a commercial design tool. By demonstration of our translator, we hope to elicit interest from potential users who would benefit from such an automated path from design notations to formal analysis, and to engage in more technical interchanges with researchers pursuing similar objectives.

ACKNOWLEDGMENTS

We are especially grateful to Gerard Holzmann, Erich Mikk and Diego Latella, all of whom have been generous with inspiration and helpful advice.

The research and development described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space administration. Funding was provided under NASA's Code Q Software Program Center Initiative UPN #323-098-5. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States government or the Jet Propulsion Laboratory, California Institute of Technology.

REFERENCES

[Day, 1993] N. Day, "A Model Checker for Statecharts (Linking CASE tools with Formal Methods)". Technical Report 93-35, Integrated System Design Laboratory, Dept. of Computer Science, University of British Columbia, October 1993.

[Czerny & Heimdahl, 1998] B.J. Czerny & M.P.E. Heimdahl, "Automated Integrative Analysis of State-Based Requirements". *Proceedings of the 13th IEEE International Conference on Automated Software Engineering*, Honolulu, Hawaii, Oct. 1998.

[Harel, 1987] D. Harel, "Statecharts: A visual formalism for complex systems." *Science of Computer Programming* 8(3):231-274, 1987.

[Harel & Naamad, 1996] D. Harel & A. Naamad, "The STATEMATE Semantics of Statecharts." *ACM Transactions on Software Engineering and Methodology* 5(4):293-333, Oct 1996.

[Heitmeyer et al, 1998] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer & R. Bharadwaj, "Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications." *IEEE Transactions on Software Engineering* 24(11):927-948, November 1998.

[Holzmann, 1997] G.J. Holzmann, "The Model Checker SPIN." *IEEE Transactions on Software Engineering*, 23(5):279-295, May 1997.

[Lattela et al, 1999] D. Latella, I. Majzik & M. Masink, "Towards a Formal Operational Semantics of UML Statechart Diagrams." To appear in FMOODS 1999.

[Lowry et al, 1997] M. Lowry, K. Havelund & J. Penix, "Verification and Validation of AI Systems that Control Deep-Space Spacecraft". Slightly revised version of a paper that appeared in Foundations of Intelligent Systems, (Eds. Z.W. Ras, A. Skowron), Tenth International Symposium on Methodologies for Intelligent Systems, Charlotte, North Carolina, Oct. 15-18, 1997. Springer-Verlag Lecture Notes in Artificial Intelligence, Vol. 1325.

[Mikk et al, 1998] E. Mikk, Y. Lakhnech, M. Siegel, G.J. Holzmann, "Implementing Statecharts in Promela/SPIN". *Proceedings, Workshop on Industrial Strength Formal Techniques*, Boca Raton, Florida, Oct. 1998.

[Schneider et al, 1998] F. Schneider, S.M. Easterbrook, J.R. Callahan & G.J.Holzmann, "Validating Requirements for Fault Tolerant Systems using Model Checking." International Conference on Requirements Engineering, 1998.