VECS - Verification Environment for Critical Systems: Tool Supported Formal Modeling and Verification

Tim Gonschorek, Marco Filax, Michael Lipaczewski, and Frank Ortmeier

Chair of Software Engineering
Otto-von-Guericke-University Magdeburg
{tim.gonschorek, marco.filax, michael.lipaczewski, frank.ortmeier}@ovgu.de

A Verification Environment for Critical Systems

In modern system engineering processes and especially for safety critical applications, formal methods get more and more in the focus of interest, because they can help to obtain special safety assurances, e.g. whether a hazardous system behavior can occur and if so, how probable it is. Although using formal methods can be helpful in the domain of systems engineering, it is not applied frequently yet. One reason, among others, is the high barrier to entry for using formal methods. On the one hand it requires the knowledge of a specific modeling language like nuXmv[1], PRISM[2] or UPPAAL[3] and on the other hand the systems engineer must be familiar with the theory of logic, e.g. for understanding and creating system specifications. Exactly that problem overcomes the verification tool VECS[4] (Verification Environment for Critical Systems)[4] (former S^3E [3]), which provides an eclipse-based IDE and verification environment for the tool-independent formal system language SAML (System Analysis and Modeling Language)[2]. SAML is mentioned as an intermediate, automata based language between arbitrary high level engineering languages, e.g. SCADE, UML or Modelica, and languages of verification tools like NuSMV, PRISM or UPPAAL (see Figure 1). VECS is an eclipse-based SAML IDE, providing features like auto-completion, syntax validation, quick-fixing, an outline module and an component view representing all state machines and there connections between each other. Moreover, it provides features specifically for automata based verification languages. VECS provides a connector to the model checkers NuSMV and PRISM, i.e., they can be called from the VECS IDE and there outputs are collected, processed and shown in a result view in VECS. This is very helpful for interpreting counterexamples, which are

[1] https://nuxmv.fbk.eu/
[2] http://www.prismmodelchecker.org/
[3] http://www.uppaal.org/
[4] https://cse.cs.ovgu.de/vecs/

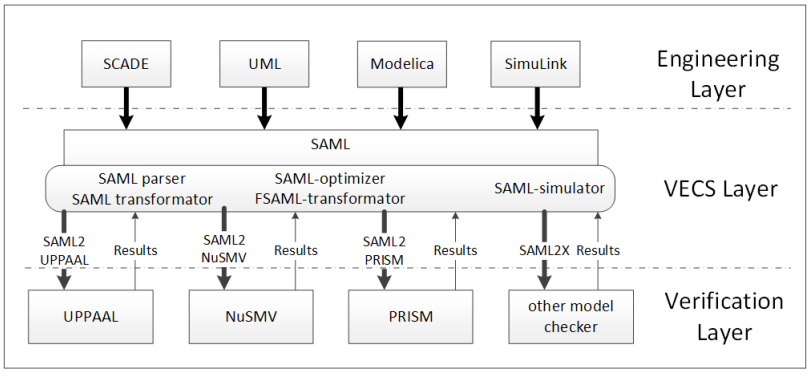

Fig. 1. SAML between engineering languages and model checking tools.

returned by a model checker, in case a witness of a hazardous state can be found. In VECS, they are shown as a structured table supporting the understanding of the result. Additionally, the VECS IDE provides an interactive debugger and simulator. It can be used to go step by step through the produced model to resolve all executed transitions as well as reached system states. Moreover, it is possible to step into a state producing a counterexample and trace backward through all corresponding transitions. This can support the understanding of the system behavior as well as the conditions leading into an hazardous state.

Since model checking is a very resource intensive task, the VECS system provides the opportunity for transferring the model checking task to a remote server system. Further, we provide a generator for Deductive Cause-Consequence Analysis (DCCA) [1] as well as a fault tree generator for the analysis of hazardous system behavior.

In summary, VECS has been developed for encouraging system engineers on using formal methods within system modeling and development, especially for the safety critical domain. It does not only support the creating of SAML automata, but VECS also supports the user in verifying the automata, understanding the verification results and improving the system's structure.

References

1. W. Reif F. Ortmeier and G. Schellhorn. Deductive cause-consequence analysis (dcca). In *Proceedings of the 16th IFAC World Congress*. Elsevier, 2005.
2. M. Güdemann. *Qualitative and quantitative formal model-based safety analysis : push the safety button*. PhD thesis, Otto-von-Guericke-University Magdeburg, 2011.
3. M. Güdemann and F. Ortmeier. A Framework for Qualitative and Quantitative Model-Based Safety Analysis. In *Proceedings of the 12th High Assurance System Engineering Symposium (HASE 2010)*, 2010.
4. M. Lipaczewski, S. Struck, and F. Ortmeier. Using Tool-Supported Model Based Safety Analysis - Progress and Experiences in SAML Development. In *Proceeding of the IEEE 14th International Symposium on High-Assurance Systems Engineering (HASE 2012)*, 2012.