Teaching and Training Formal Methods for Safety Critical Systems

Michael Lipaczewski and Frank Ortmeier
Computer Systems in Engineering
Otto-von-Guericke University Magdeburg
{michael.lipaczewski,frank.ortmeier}@ovgu.de

Abstract—Embedded systems become a major part in many domains. This also involves systems which might create heavy damages and injuries when they fail. However, because of the rising number of software components used within this embedded hardware, safety-related problems are hard to discover, and it is even harder to prove that there are none.

One approach to guarantee the correctness of a system is model-based safety analysis. They rely on an abstract representation of the system which can then be analyzed using model checkers. The results of these analysis are in general much more precise and often reveal surprising results of failure combinations, where no one had ever thought about before.

Nevertheless model-based safety analysis is not used widely. Mainly because it is not well-known and hard to apply to current safety standards which rely on manual approaches. Another fact might be, that most approaches are scientific and in most cases prototypes that are hard to use.

In this paper we present some ideas and first steps towards an easy to learn and easy to use model based safety approach. Additionally we present different user-interfaces that are supposed to support the user in his learning.

Keywords—Model-based, Safety Analysis

I. INTRODUCTION

In almost all domains, we rely on technology to make lives easier and more comfortable. To achieve this goal, technology not only have to become smarter, but also smaller. Therefore, it is necessary to replace hardware controls with software components that don’t need any space besides a processing unit, and that can handle much more functionality. These systems are called cyber-physical systems.

However, these systems are also used in safety-critical applications, where a failing system can endanger human lifes. Traditional safety analysis relies on hardware-based controllers, which are easy to predict. For software components, behavior is much harder to predict, and can become non-deterministic due to bugs. To cope with this kind of systems, model-based safety analysis approaches were created that are capable of modelling and analysing non-deterministic behavior.

Yet, even though model-based approaches exist and model checkers are improving, they are barely used in the industry for safety-critical systems. One reason is that they are not well-known and difficult to apply. Another handicap is that they have not yet been practically shown to be at least as good as the approaches that are currently in use. Additionally, most of these tools have a steep learning curve, requiring hours to correctly set up the necessary toolchain, and days of studying before they can be used productively.

In earlier work, [1], [2], we presented a language and methodology to easily model software and hardware components as well as environmental dependencies and potential malfunctions of the system. We used this language to prove the correctness of the system as well as giving a complete list of minimal failure sets that will lead to a hazardous malfunction of the whole system [3]. We also extended this language to make modeling easier [4] and to add additional types of model analysis [5].

To train students and engineers to successfully use model-based approaches, a lot of theoretical background, tools and knowledge is needed. This involves a lot of effort before a first model can be created and checked. To improve motivation and knowledge retention, more rapid results are needed. To reduce the set of required knowledge, we simplify the modeling and verification process and hide the tool-layer from the user. The idea was to create a user-interface that supports an engineer in using model-based approaches without requiring in-depth knowledge of the underlying system.

To that end, our UI provides code completion and instant code verification, as well as integration of the toolchain into the IDE. We created two different user interfaces with different abilities and target audiences. In this paper we are going to discuss the differences and scope of these two development environments.

In the following sections we present the basic Safety Analysis Modeling Language (SAML) along with language extensions that help engineers to successfully build their models (Section II). Section III introduces the Eclipse-based user interface, which not only provides editor-support, but also integrates the underlying toolchain. To allow potential users to gain hands-on experience without having to install the software, we also built a web-based editor with tutorials and example models. This UI will be presented in Section IV. Section V compares the two different user interfaces with respect to their educational use. Finally, SAML can be used in non safety-critical settings as well. These non safety-related SAML extensions are explained in Section VI. Current and further work to gain more user acceptance are presented in Section VII followed by the conclusion in Section VIII.
II. System Analysis Modeling Language

The System Analysis Modeling Language (SAML) was created as an intermediate language for high-level engineering tools and model checkers. The main benefit of this language proven semantic-preserving transformations into several different model checkers input languages, which can be used to analyze different aspects of a model using third-party tools.

The language is used to model a set of time-discrete, synchronous finite state automata. Meaning that all automata compute their state for the next time step in parallel and only based on the previous system state. The main element of a SAML model is a component, which contains a number of states variables – each defined by its integer range –, and a number of update rules. Update rules consists of conditions (boolean expressions) that, when fulfilled, cause their respective assignment rules to be executed. An assignment rule can contain multiple non-deterministic expressions, which in turn can each contain a probabilistic distribution with which the automaton will go into a specified next state.

Depending on which type of assignment rule is used, it is possible to express deterministic decisions (mostly used for functional behavior), probabilistic choices (e.g. to model hardware failures) and non-determinism (e.g. to model software failures with unknown probabilities).

Additionally, several extensions were made to the language to be able to create models even simpler and faster:

- **Constants** are integer or double-valued variables. They can be used to group and organize probabilities, enable model parameterization, and reduce redundancies.
- **Formulas** are single-expression functions. They can be of boolean or double type and are useful to make models smaller and better readable.
- **Sub-components** are used to organize components, constants and formulas hierarchically.
- **Enumerations** also improve readability and error-freeness of models by using sets of symbolic constants instead of integer ranges. Especially for larger models, it is much easier to name the state than to find the integer that represents the desired state.
- **Includes** allow to build models spreading over several files. With this features, it is possible to create model libraries with components that are reused in several projects.
- **Failure Components** are a special type of component representing common failure occurrence patterns. These include persistent and transient failures with per-time or per-demand failure occurrences. As using failure components semantically marks the component as modelling failure occurrence patterns, algorithms that determine minimal critical failure sets can directly work on the set of failure components.
- **Templates** are used to efficiently create multiple component instances that differ only slightly from each other.

All extensions aim at making the modeling of a system easier and faster, as well as improving the readability of the model. However, they do not increase the expressiveness of SAML: as all extensions can be transformed into a simple SAML model containing only components with update rules and assignment rules. Thus, a core SAML language exists that is easy to learn and use because of its simplicity. And for experienced users, advanced features can be learned step by step to enable them to model their systems more efficiently and with less errors. With this approach, trainees can succeed early without having to know the full SAML language.

III. VECS Framework

We created a complete modeling and verification environment called VECS (Verification Environment for Critical Systems), where we tried to implement all features that are already used in development environments for modern programming languages.

VECS allows for the execution of the third-party backend model checker for systems modelled in SAML through its IDE. Thus, the underlying model checker is hidden from the user, reducing the knowledge required to use model-based safety analysis. VECS automatically transforms the user’s SAML model into the target language, runs the according model checker and parse the results, which are then displayed inside the IDE. This integration was done using the Eclipse framework, which additionally provides a simple installation and update process, as well as a familiar and widely used window management. A screenshot of the current VECS version is given in Figure 1.

Additionally, we extended the Eclipse-based SAML editor with feature such as syntax highlighting, auto-completion, auto-formatting and instant error-checking for the SAML language - features that users know from Eclipse-based editors for other programming languages. The Eclipse module outline pane and project manager are also available in VECS. The results of backend model checkers are parsed and displayed in easily readable views. Additionally, the model behavior can be simulated step by step in order to analyze and understand the cause of unexpected or incorrect behavior.

Besides the user interface, the VECS framework also contains a console input for automation purposes as well as a client-server architecture for the backend model checkers that allows for the transparent execution of the model checking process on a different computer. The main intention for this remote model checking was the huge amount of RAM and time that is needed for model checking on a desktop computer. But it also enables different SAML editor frontends to use the VECS model transformation and model checking infrastructure. One of these other frontends is presented in the next section.

IV. Web-Based Tutorials

Another approach for even quicker introduction to SAML is our web frontend tryسامل.org. It does not have (and is not
intended to have) all features of the full Eclipse IDE, but allows users to try out SAML without having to install any software on their computers. It also included tutorials and a SAML language reference for users not familiar with that language.

The main purpose of the web editor is to familiarize interested users with the language and the results that can be generated using our approach. For model-checking, the user model is sent to our server, where it is checked using the client-server architecture of the VECS framework. As we cannot offer unlimited amount of computation power, the web editor has some limitations regarding the size of the model as well as the number of people who can use the model-checking feature simultaneously. Nevertheless, it is an easy approach to learn the language and its abilities before taking the time to set up an own SAML environment.

Besides the pure editor with syntax highlighting, trysaml.org contains a beginner and an advanced tutorial to explain the different SAML elements. The beginner tutorial mainly focuses on the three main SAML concepts (components, update rules, assignment rules) and their usage. It also gives a first insight into the different types of model checkers that can be used. The advanced tutorial then introduces more language features. The web frontend also includes a number of example models that can be viewed and edited inside the browser.

Figure 1 shows one step in the beginner tutorial. The editor view is on the right-hand side, while the tutorial text is displayed to the left of the editor. The current tutorial task is given above the editor. For model checking, the bottom part offers two different model checkers as well as an input field for the property to be validated.

After completing the tutorial, we encourage users to download and install the Eclipse-based VECS IDE for a faster and easier way to create and verify their models.

V. COMPARISON OF USER INTERFACES REGARDING EDUCATION

Both editors fulfill different needs of trainees. The web-based version allows for quick results and self study, while the Eclipse version can cope with bigger models and offers more assistance to the user. Table I shows a comparison of the two user interfaces with respect to teaching and usability:

Software requirements that are needed to run the environment. The browser-based approach requires only a generic recent web browser, which is pre-installed on all
Eclipse-based software requirements – +
integrated documentation – +
interactive tutorial – +
prior knowledge – +
learning feedback – +
# concurrent users + –
editor support + –
static model checks + –
offline mode + –
platform independence – +

browser-based

Table I
COMPARISON OF THE DIFFERENT DEVELOPMENT ENVIRONMENTS FOR SAML

consumer computers. For the Eclipse based approach, manual installation of Eclipse itself, the VECS Eclipse plugin, and additional third-party model checkers is necessary.

Integrated documentation is included on the website, but not shipped with the Eclipse plugin. As the target audience of the eclipse plugin are experienced SAML users, documentation is rarely needed.

Interactive tutorial is present on the website and is actually the reason why the web interface was created. Therefore self-study is possible using the web interface. In contrast, in order to work with the Eclipse version, users already have to understand the concepts behind SAML (e.g. by first completing the online tutorial). So the Eclipse version may only be used for learning SAML when an experienced tutor is present.

Prior knowledge that is necessary or recommended to use the editor. Because the online version contains a language reference as well as a tutorial, no knowledge is needed, whereas the VECS IDE requires knowledge about the syntax of SAML, to be able to use it successfully.

Learning feedback is mainly given by the web-based system, where the completion of a tutorial task by the user is recognized. For the Eclipse version, some learning effect can be achieved by the static model checks, where the user is informed about errors in the model. Nevertheless, as the Eclipse editor is not supposed to teach there is little feedback suitable for students.

Number of concurrent users describes the number of people that can use the system at the same time. As the Eclipse IDE and its model checkers run directly on the user’s computer, the number of concurrent users here is not limited. On the other side, because of server limitations, the number of people who can use the web-based system at the same time is severely limited. While the model syntax...
checks are performed in the browser, model transformation and backend model checkers are executed on our limited number of servers, and are currently limited to about 30 concurrent model checks.

**Editor support** is given in a limited way by the web frontend, which includes only syntax highlighting and no other advanced features. The Eclipse version comes with full support for auto-completion, quick-fixes, auto-formatting and a lot of other features known from modern development environments.

**Static model checks** are one of the major features of the Eclipse version. Besides model syntax checks, some semantic checks can be performed (e.g. whether the probabilities for specified alternative behavior sum up to one). In the web-based version, only the syntax can be checked.

**Offline mode** can only be supported by the Eclipse based system, as the browser-based approach requires internet access to our backend servers for model checking.

**Platform independence** is given by both approaches. Eclipse and VECS both using Java, and thus are usable on a variety of platforms. A limiting factor is the availability of the backend model checkers for the different platforms. The model checkers currently in use for VECS are available for Windows, Linux and MacOS. The web frontend only requires a recent web browser, but no additional plugins. Thus, it cannot only be used on the three platforms where VECS is available, but also on other systems such as Solaris workstations and even smartphones and tablets.

Overall, the browser-based version is mainly intended for self-study and to try out SAML, whereas VECS’ application areas is to support the user in every day use. But it can also be used for teaching along with a tutor.

VI. **NON SAFETY-RELATED APPLICATIONS FOR SAML**

Besides safety-related application areas, VECS can also be used to verify other system properties for other types of engineering problems. One of these areas is functional correctness, which is easy to check using qualitative model checkers, which VECS uses for the computation of minimal critical failure sets anyway.

Another application area is the optimization of models with respect to user-specified objective functions. Special keywords in SAML mark constants that may be varied to yield different model instances. The VECS framework is then able to determine an optimal model instance with respect to the given objective function [6].

VII. **FUTURE WORKS**

Currently, we work on improving the user experience. Through a user study, several possible improvements on ease of modelling were identified. The most important planned improvement is the integration of a satisfiability checker to determine missing or overspecified conditions within the assignment rules at modeling time. Additionally, a graphical result viewer is planned to improve spotting and understanding of interesting aspects of model checking results.

Another big topic is the graphical representation of the SAML model. In a first step, export to a graphical representation is planned, to visualize an existing model. This requires a definition of graphical elements that represent the single elements of a model. In a second step, a full graphical editor is planned. But for that goal, several problems have to be faced: Graphical models quickly grow unintelligible with increasing models size. Therefore it can be used only some models, and a textual editor is still required for models containing many states or complex arithmetic expressions.

In this context, a model import facility from UML modelling tools (e.g. Rhapsody) might also be useful. This would allow for the import of existing UML models in order to perform safety-related verification in VECS. This would also improve user acceptance for users with pre-existing UML models, as it helps them to not have to two maintain redundant models.

VIII. **CONCLUSION**

In this paper we presented the formal model specification language SAML and the VECS framework that transforms SAML models into the input language of several model checkers. Two different user interfaces were presented, each with its own advantages and disadvantages. We analyzed this development environments regarding their usability for educational purposes.

The presented web-based version is a good choice for people who want to gain first experiences with SAML without having to install any software. It also offers online tutorials and a language reference, and can be used for self-study.

The VECS Eclipse-based editor offers less self-studying abilities, but comes with advanced IDE features that are known from modern programming languages.

Both frontends hide the integrated backend model checkers from the user to reduce the knowledge needed to use our model-based approach.

Additionally we presented strategies to improve the user experience as well as extending our safety-related approach towards a more general engineering tool.

ACKNOWLEDGMENT

Michael Lipaczewski is sponsored by the Deutsche Ministerium für Bildung und Forschung in the ViERforES project (BMBF, project-Nr.: 01IM08003C).

The presented software VECS can be downloaded from the website of the research group: [http://cse.cs.ovgu.de](http://cse.cs.ovgu.de) - Research - Software. The online editor can be visited and used under [http://www.trysaml.org](http://www.trysaml.org).
REFERENCES


