Tools for Critical Systems Development with UML (Tool Demo)

Jan Jürjens* and Pasha Shabalin

Software & Systems Engineering, Dep. of Informatics, TU Munich, Germany

Abstract. The high quality development of critical systems (be it dependable, security-critical, real-time, or performance-critical systems) is difficult. Many critical systems are developed, deployed, and used that do not satisfy their criticality requirements, sometimes with spectacular failures. UML offers an opportunity for high-quality critical systems development that is feasible in an industrial context, if tools can be provided which automatically check important criticality requirements.
We present research on developing tool-support for critical systems development with UML. The developed tools can be used to check the constraints associated with UML stereotypes representing criticality requirements mechanically, based on XMI output of the diagrams from the UML drawing tool in use. We also explain a framework for implementing verification routines for the constraints associated with such stereotypes. The goal is that advanced users of the CSDUML approach should be able to use this framework to implement verification routines for the constraints of self-defined stereotypes.

Introduction This article presents tool support for the automated analysis of UMLsec models with regard to security requirements developed at TU Munich and available at [JSA$^+$04]. It also describes a framework which allows inclusion of analysis plugins for other non-functional properties, such as dependability requirements. More details, including references to related work, can be found in [JS04,Jür04].

Background on UMLsec We give some background on the UML extension for secure systems development called UMLsec: Recurring security requirements (such as secrecy, integrity, and authenticity) are offered as specification elements by the UMLsec extension. The properties are used to evaluate diagrams of various kinds and to indicate possible vulnerabilities. One can thus verify that the stated security requirements, if fulfilled, enforce a given security policy. One can also ensure that the requirements are actually met by the given UML specification of the system. UMLsec encapsulates knowledge on prudent security engineering and thereby makes it available to developers who may not be experts in security. The extension is given in form of a UML profile using the standard

* http://www4.in.tum.de/~juerjens – Supported within the Verisoft Project of the German Ministry for Education and Research.

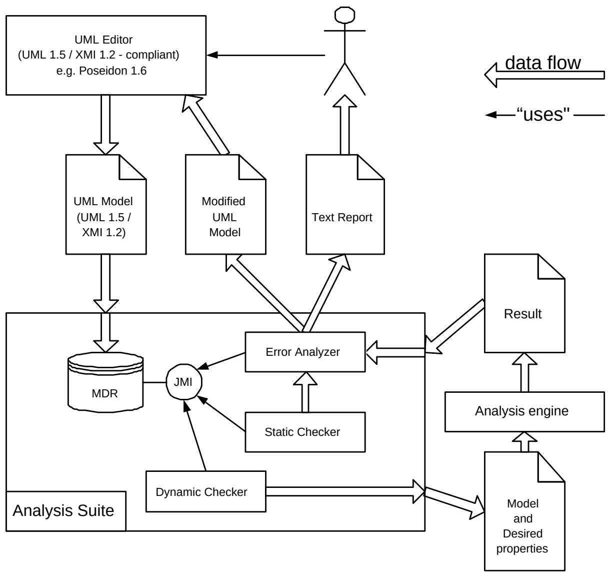


Fig. 1. UML tools suite

UML extension mechanisms. *Stereotypes* are used together with *tags* to formulate security requirements and assumptions on the system environment. *Constraints* give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics mentioned below. The extension has been developed based on experiences on the model-based development of security-critical systems in industrial projects involving German government agencies and major banks, insurance companies, smart card and car manufacturers, and other companies. Note that an extension of UML to an application domain such as security-critical systems that aims to include requirements from that application domain as stereotypes, as opposed to just adding specific architectural primitives, can probably never be fully complete. We expect UMLsec to be extended with additional, more specific concepts (for example, from more specialized application domains such as mobile security).

The Framework The architecture and basic functionality of the UMLsec analysis suite are illustrated in Fig. 1. The overall architecture is divided between the UML drawing tool in use and the analysis suite. This way the analysis suite can be offered as a web application, where the users use their drawing tools to construct the UML model which is then uploaded to the analysis suite. Additionally, a locally installable version is available. The usage of the analysis suite as illustrated in Fig. 1 proceeds as follows. The developer creates a model and stores it in the UML 1.5/XMI 1.2 file format.[1] The file is imported by the tool

[1] An upgrade to UML 2.0 is in development.

into the internal MDR repository. The tool accesses the model through the JMI interfaces generated by the MDR library. There are static checkers that parse the model, verify static features, and deliver the results to the error analyzer. Various dynamic checkers translate the relevant fragments of the UML model into the input language of several analysis engines (including model-checker and automated theorem provers). The analysis engines are spawned by the UML suite as external processes. Their results, and possibly a counter-example in case a problem was found, are delivered back to the error analyzer. The error analyzer uses the information received from both the static checkers and dynamic checkers to produce a text report for the developer describing the problems found, and a modified UML model, where the found errors are visualized and, as far as possible, corrected.

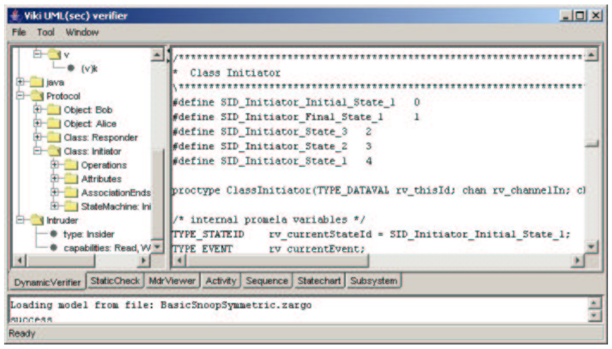
There exist various analysis plugins for the UMLsec tool framework, including:

- a tool-binding to the model-checker Spin to verify cryptographic protocols, described in [JS04],
- a tool-binding to first-order logic automated theorem provers such as e-Setheo and SPASS,
- a test-sequence generation for subsystems, sequence diagrams, activity diagrams, and statechart diagrams, and
- a checker for the static security constraints in UMLsec.

We now explain a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes. The goal is that advanced users of the UMLsec approach should be able to use this framework to implement verification routines for the constraints of self-defined stereotypes. In particular, the framework includes the UMLsec tool web interface, so that new routines are also accessible over this interface. The idea behind the framework is thus to provide a common programming framework for the developers of different verification modules which in the following we just call *tools*. Thus a tool developer should be able to concentrate on the verification logic and not be required to become involved with the input/output interface. Different tools implementing verification logic modules can be independently developed and integrated. At the time of writing, there exist verification modules for most UMLsec stereotypes. An added tool implementation needs to obey the following assumptions:

- It is given a default UML model to operate on. It may load further models if necessary.
- The tool exposes a set of commands which it can execute.
- Every single command is not interactive. They receive parameters, execute, and deliver feedback.
- The tool can have an internal state which is preserved between commands.
- Each time the tool is called with a UML model, it may give back a text report and also a UML model.

These assumptions were made in order for the framework to cover as much common functionality as possible while not becoming overly complicated. Ex-



```
/*****************************************************************
*  Class Initiator
\*****************************************************************
#define SID_Initiator_Initial_State_1   0
#define SID_Initiator_Final_State_1     1
#define SID_Initiator_State_3   2
#define SID_Initiator_State_2   3
#define SID_Initiator_State_1   4

proctype ClassInitiator(TYPE_DATAVAL rv_thisId; chan rv_channelIn; cl

/* internal promela variables */
TYPE_STATEID    rv_currentStateId = SID_Initiator_Initial_State_1;
TYPE_EVENT      rv_currentEvent;
```

```
Loading model from file: BasicSnoopSymmetric.zargo
success
```

Fig. 2. UML verification framework: screenshot

perience indicates that the assumptions are not too restrictive, given the architecture in Fig. 1. The tool architecture then allows the development of the verification logic independently of the input and output media with minimum effort. Each tool is required to implement the ITextMode interface which exposes tool functionality in text mode, with a string array as input and text as output. The framework provides default wrappers for the graphical user interface (GUI) GuiWrapper and the web mode WebWrapper. These wrappers enable use of the tool without modifications in the GUI application which is part of the framework, or through a web interface by rendering the output text on the respective media. However, each tool may itself implement the IGuiMode and/or IWebMode to fully exploit the functionality of the corresponding media, for example to fully use GUI mode capabilities to display graphical information. The GUI is shown in Fig. 2.

Future Work We plan to extend the framework with a functionality which allows advanced users to conveniently add self-defined stereotypes with tags and constraints to the tool-support.

References

JS04. J. Jürjens and P. Shabalin. Automated verification of UMLsec models for security requirements. In J.-M. Jézéquel, H. Hußmann, and S. Cook, editors, *UML 2004 – The Unified Modeling Language*, volume 2460 of *LNCS*, pages 412–425. Springer, 2004.

JSA$^+$04. J. Jürjens, P. Shabalin, E. Alter, A. Gilg, S. Höhn, D. Kopjev, M. Lehrhuber, S. Meng, M. Schwaiger, G. Kokavecz, S. Schwarzmüller, and S. Shen. UMLsec tool, 2004. Accessible through a webinterface via [Jür04]. Available as open-source.

Jür04. J. Jürjens. UMLsec webpage, 2002-04. Accessible at http://www.umlsec.org.

Jür04. J. Jürjens. *Secure Systems Development with UML*. Springer, 2004.