Tools for Secure Systems Development with UML: Security Analysis with ATPs

Jan Jürjens* and Pasha Shabalin

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

Abstract. We present tool-support for checking the security requirements associated with UMLsec stereotypes. A framework supports implementing verification routines, based on XMI output of the diagrams from UML CASE tools. Advanced users of the UMLsec approach can use this open-source framework to implement verification routines for the constraints of self-defined stereotypes. We focus on a verification routine that automatically verifies sequence diagrams with cryptographic algorithms for security requirements by using automated theorem provers.

The analysis suite for UMLsec [Jür04] models available at [UML04] is illustrated in Fig. 1. The developer creates a UML 1.5 model and stores it in the XMI 1.2 file format (an upgrade to UML 2.0 is in development). Note that some UML CASE tools do not implement XMI correctly, in which case one might have to correct the format for example with a script. The file is imported into the tool's repository, using the data-binding framework MDR. By using MDR, the framework can handle all UML constructs for which a translation to XMI exists in the relevant DTDs released by the OMG. The tool accesses the model through the JMI interfaces generated by the MDR library. Static checkers parse the model and verify it directly for static requirements. Dynamic checkers translate the relevant fragments of the UML model into the input language of several analysis engines (such as model-checkers and automated theorem provers). That way, the UML models can be analyzed for dynamic requirements, which may be formulated in temporal logic, or potentially using OCL. 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. For the dynamic checkers, a reference semantics for a simplified fragment of UML exists in [Jür04], which is however not enforced by the framework but at the responsibility of the tool developer, as well as achieving semantic consistency between different tools. 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 currently exist various analysis plugins for the UMLsec tool framework, including:

* `http://www4.in.tum.de/~juerjens` . This work was partially funded by the German Federal Ministry of Education, Science, Research and Technology (BMBF) in the framework of the Verisoft project under grant 01 IS C38. The responsibility for this article lies with the author(s).

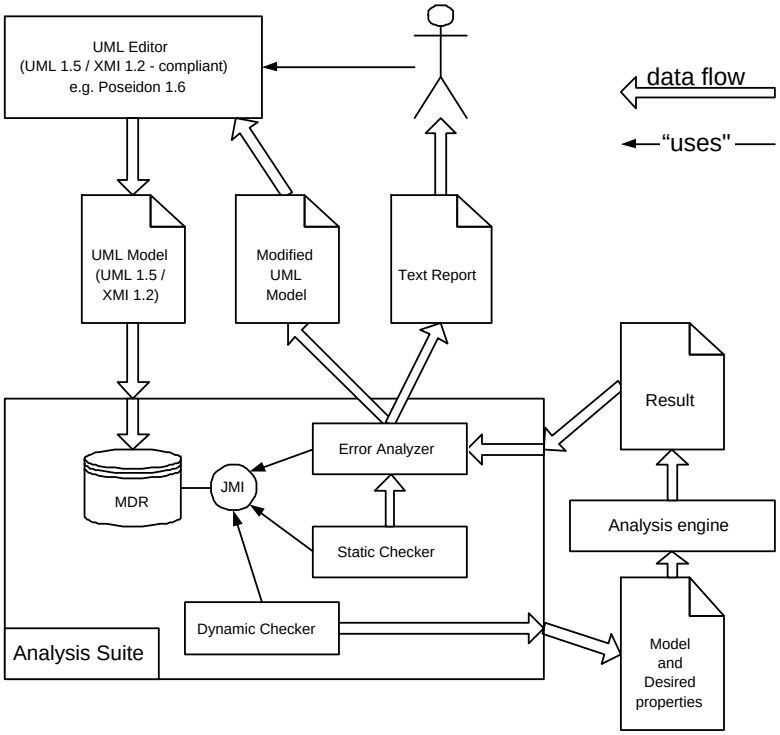
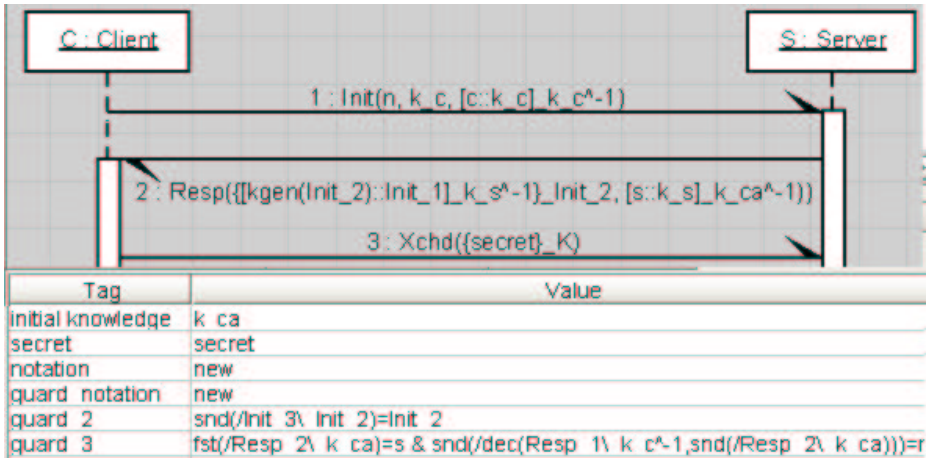


Fig. 1. UML tools suite

- a tool-binding to the model-checker Spin to verify cryptographic protocols,
- a tool-binding to first-order logic (FOL) automated theorem provers,
- a test-sequence generation for subsystems, sequence diagrams, activity diagrams, and statechart diagrams, and
- a checker for the static security constraints in UMLsec.

Advanced users of the UMLsec approach can use this framework to implement tools for constraints of self-defined stereotypes. The developer can concentrate on the verification and need not become involved with the input/output interface. A tool only needs to obey the following assumptions made to keep framework and tools simple but retain as much functionality as possible:

- It is given a UML model as input and may load further models if necessary.
- The tool exposes a set of commands which it can execute.
- A command is non-interactive. It receives parameters, executes, and returns its output.
- Each time the tool is called with a UML model, it may give back a text report and also a UML model.
- The tool can execute several commands consequently; the internal state of the MDR repository and all tools is preserved between command calls.



Tag	Value
initial knowledge	k_ca
secret	secret
notation	new
guard_notation	new
guard_2	snd(/Init_3_Init_2)=Init_2
guard_3	fst(/Resp_2_k_ca)=s & snd(/dec(Resp_1_k_c^-1,snd(/Resp_2_k_ca)))=r

Fig. 2. Example

– The set of commands available for each tool may vary depending on the execution history and current state.

On any Java-enabled platform, the tool can run as a console application (interactive or batch mode), a Java Servlet on a webserver, and a GUI application executed locally. For this, each tool integrated in the UML framework must only implement one common interface. We now focus on a tool which automatically verifies sequence diagrams including cryptographic algorithms for security requirements by using automated theorem provers.

Sequence Diagram Analyzer using ATPs The sequence diagram to be analyzed is drawn using a UML CASE tool. The analyzer produces an abstract interpretation of the execution semantics of the diagram, and the security requirement to be verified, as a FOL formula in the TPTP format. TPTP is an input notation used by many automated FOL theorem provers such as e-Setheo and SPASS. More information about the security analysis method can be found in [Jür05]. Here we concentrate on the tool issues. A more comprehensive tutorial can be found at [UML04]. The following notation is supported for cryptographic algorithms: The encryption of the expression E under the key K is written as $\{E\}_K$, the decryption of E using K as $<E>_K$, the signature of E using K as $[E]_K$, the extraction of the signature E using the verification key K as $/E\backslash_K$, and the private key belonging to the public key K is written as K^{-1}. To use an argument in another message or guard, one can make use of the variables in which incoming values are stored. Each variable is named by the name of the operation which it is the argument of, followed by the number of the position of the argument. For example, Init_5 is the 5th argument of the message with the function-name Init. An example drawn in the UML tool Poseidon is shown in Fig. 2. Tagged values can be used to attach additional information to be used in the analysis:

Attacker's initial knowledge The attacker's initial knowledge is stored in a tag *initial knowledge*. One tag is defined for each such value.

Attack If there is a tag *secret* with a value *value*, the security conjecture is generated in TPTP which checks whether the data item *value* will remain secret against the attacker considered during execution of the diagram. Alternatively, the TPTP conjecture can be stored in the tag *conjecture*.

Message ordering With the tag *order*, one can determine whether the implementation of the sequence diagram enforces the message ordering at the receipt of messages (which is the standard UML semantics for sequence diagrams), or not (which is what is implemented for example at many smartcards, see [Jür05]. By default, the order is respected.

Variable notation With the tag *notation*, one may switch between different ways of defininining the variable names that store the incoming arguments.

Guard notation For UML CASE tools which do not directly support the use of guards in sequence diagrams (such as Poseidon 1.6), one can include them in front of the stimulus labels, or as tagged values with the tag-name *guard_NR*, where *NR* is the number of the stimulus in the diagram to which the guard belongs. The tag may be defined at any model element in the diagram. Using the tag *guard_notation* one can switch between these two alternatives.

Facts First-order formulas in the TPTP notation can be added as axioms to the TPTP file by storing them in tagged values with the tag-name *fact*.

Related Work There seems to be no work yet on connecting ATPs to UML CASE tools. Work on providing security analysis tool support for UML is performed by the DEGAS project [DEG01]. More generally, there have been a number of approaches for tool-support verifying general properties of UML diagrams, mostly by connecting UML CASE tools to model-checkers, including [LP99,SKM01].

Conclusion The framework has been used in several industrial applications: for example, the binding to the automated theorem prover e-Setheo has been used to verify the Common Electronic Purse Specifications and a biometric authentication system. Experiences have been favorable (see [Jür05]).

Acknowledgements Fruitful collaboration with the UMLsec group members, especially Andreas Gilg, on implementation issues is gratefully acknowledged.

References

[DEG01] Degas, 2001. http://www.omnys.it/degas.

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

[Jür05] J. Jürjens. Sound methods and effective tools for model-based security engineering with UML. In *ICSE 2005.* IEEE Computer Society, 2005.

[LP99] J. Lilius and I. Porres. Formalising UML state machines for model checking. In *UML 1999*, volume 1723 of *LNCS*, pages 430–445. Springer, 1999.

[SKM01] T. Schäfer, A. Knapp, and S. Merz. Model checking UML state machines and collaborations. In S.D. Stoller and W. Visser, editors, *Software Model Checking*, volume 55 of *ENTCS*. Elsevier, 2001.

[UML04] UMLsec tool, 2002-04. Open-source. Accessible at http://www.umlsec.org.