Developing Security-Critical Applications with UMLsec
A Short Walk-Through

Jan Jürjens

Software & Systems Engineering, TU München, Germany,
http://www4.in.tum.de/˜juerjens

Abstract. Developing high-assurance security-critical systems is difficult and there are many well-known examples of security weaknesses exploited in practice. Thus a sound methodology supporting secure systems development is urgently needed. We give an overview over an extension of UML, called UMLsec, that allows expressing security-relevant information within the diagrams in a system specification. We present tool-support which has been developed for the UMLsec approach. We explain the methodology at the hand of a simple example.
Keywords. Secure software engineering, security engineering, security verification, formal methods in security, security evaluation, security models, cryptographic protocols.

1 Motivation

The high quality development of critical systems (be it dependable, security-critical, real-time, performance-critical, or hybrid systems) is difficult. Many critical systems are developed, deployed, and used that do not satisfy their criticality requirements, sometimes with spectacular failures. Part of the difficulty of critical systems development is that correctness is often in conflict with cost. Where thorough methods of system design pose high cost through personnel training and use, they are all too often avoided.

The Unified Modeling Language (UML) offers an unprecedented opportunity for high-quality critical systems development that is feasible in an industrial context.

- As the de-facto standard in industrial modeling, a large number of developers is trained in UML.
- Compared to previous notations with a user community of comparable size, UML is relatively precisely defined.
- A number of analysis, testing, simulation, transformation and other tools are developed to assist the every-day work using UML.

This article gives a short introduction into using UML for critical systems development and to contribute to overcoming the challenges involved. We sketch how one can use stereotypes, tags, and constraints to encapsulate knowledge on prudent critical systems engineering and thereby make it available to developers which may not be specialized in critical systems. We also demonstrate how one can check whether the constraints associated with the stereotypes are fulfilled in a given specification using the tool-support provided. This way one can find flaws present in the design, before a system is deployed, or even implemented. The approach can also be used to analyze code for weaknesses using model-based criticality testing.

2 Walk-through using an Automotive Case-study

For adaption to a particular application domain UML provides three "light-weight" extension mechanisms: *Stereotypes* give a specific meaning to the model elements they are attached to and are represented by double angle brackets. A *tagged value* is a name-value pair in curly brackets associating data with elements in the model. Furthermore, *constraints* may be attached that have to be satisfied by the diagram. In this section we take a simple, fictitious case-study describing an automotive toll collection system to demonstrate our ideas.

Secure Business Processes with Activity Diagrams To start with our example of a toll collection system: to pay toll, a driver has to be registered at a toll station. He then passes several toll stations. When he leaves the road, the payment should be performed in a way that prevents either of the involved parties from cheating. In Fig. 1, this security requirement is expressed using the «fair exchange» stereotype . The actions listed in the tags {start} and {stop} should be linked in the sense that if one of the former is executed then eventually one of the latter will be (this can be checked using the automatic tool support explained below). This means that, once the driver has paid his toll, either he gets a receipt for his payment immediately, or he is able to refuse the payment at that point.

Physical Security using Deployment Diagrams When the driver approaches a toll station, a wireless connection is established. The payment transaction requires transmission of data to be kept secret. This information on the physical layer and the security requirement is reflected in the deployment diagram in Fig. 2: The stereotype «secure links» expresses the constraint that security requirements on the communication are met by the physical layer, which can again be checked automatically. In the given diagram, this constraint associated with the stereotype «secure links» is violated when considering standard adversaries, because simple wireless connections can be eavesdropped easily, and thus the

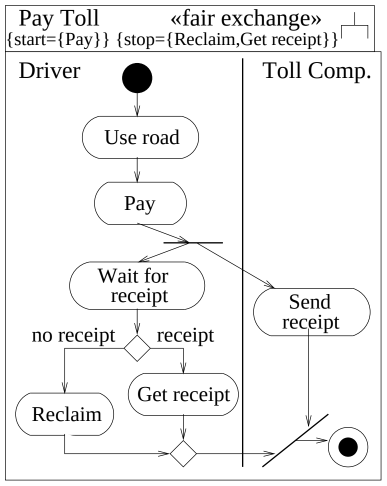

Fig. 1. Pay toll activity diagram

data that is communicated does not remain secret. For this adversary type, the stereotype is thus applied wrongly to the subsystem.

Secure states using statechart diagrams Continuing with the example, every time the drivevr passes a toll station, some data is added to each of two lists. First, the list *location* is filled with the stations that he has passed and second, the list *amount* is filled with the corresponding toll amount. We further assume that the lists are sequentially ordered in that way that the ith element in the *locations* list is assigned to the ith amount in the *amount* list. The first entry in the lists is the first toll station which has been passed, the second entry the second one and so on. In Fig. 3, we can see that a collection company only has access to this list of amounts by using the operation `ga()`. For privacy reasons we forbid the

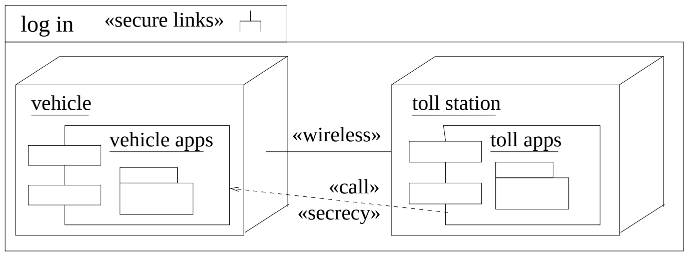

Fig. 2. Physical layer of toll application

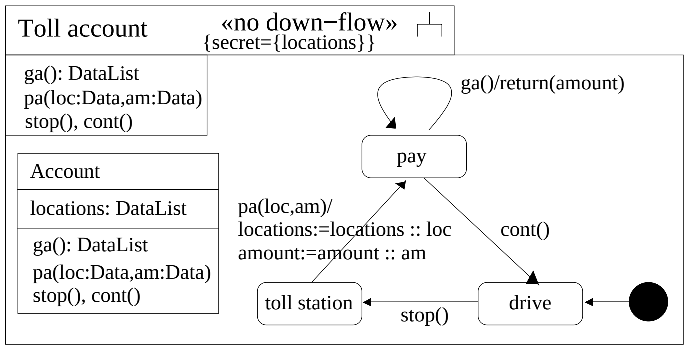

Fig. 3. Toll account data object

returning of the locations list to the company. But since the distances between the toll stations are usually different, one can derive from the toll amounts in the list the sections on the road that have been passed. Thus, the different locations are leaked out implicitly.

In the statechart diagram we use the stereotype ⟪no down − flow⟫ to indicate that the object should not leak out any information about secret data (in this case the locations). Unfortunately, the given specification violates the requirement (as can be made mathematically precise). Therefore, the model carries the stereotype illegitimately. Again, this can be checked automatically.

2.1 Tool Support

To facilitate the application of our approach in industry, automated tools for the analysis of UML models using the suggested semantics are required. We describe a framework that incorporates several such verifiers currently developed at the TU München.

The Fig. 4 illustrates the architecture of the UML tool framework which meets the listed requirements. We briefly describe its functionality. The developer creates a model and stores it in the UML 1.5 / XMI 1.2 file format. The file is imported by the tool into the internal MDR repository. The tool accesses the model through the JMI interfaces generated by the MDR library. The checker parses the model and checks the constraints associated with the stereotype. The results are delivered as a text report for the developer describing found problems, and a modified UML model, where the stereotypes whose constraints are violated are highlighted.

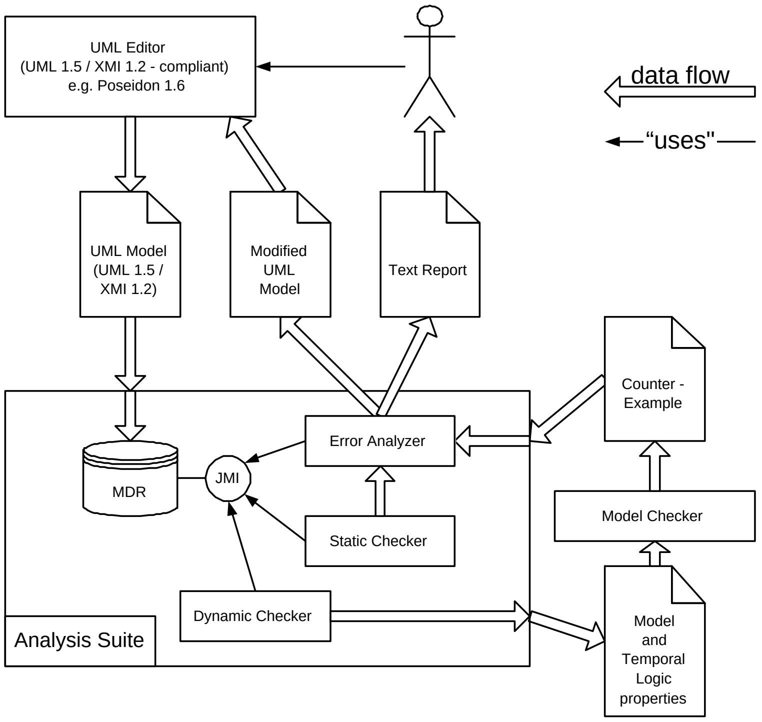

Fig. 4. UML tools suite

3 Experience and Outlook

The method proposed here has been successfully applied in critical systems projects, for example in an evaluation of the Common Electronic Purse Specifications under development by Visa International and others, in the project FairPay funded by the German Ministry of Economics, in projects with a large German bank, and in the Verisoft project funded by the German Ministry of Science and Technology. In particular, these experiences indicate that the approach is adequate for use in practice.

Given the current state of critical systems in practice, with many weaknesses reported continually, it seems to be a promising idea to apply model-driven development to critical systems, since it enables developers with little background in critical systems to make use of critical systems engineering knowledge encapsulated in a widely used design notation. Since there are many highly subtle critical requirements which can hardly be verified with the "bare eye", even critical systems experts may profit from this approach. Although the approach explained here concentrates on the flaws arising from the design level, it may be extended to analyzing code for critical weaknesses using model-based criticality testing.

For these ideas to be of benefit in practice, it is important to have intelligent tool-support to assist in using them. As sketched above, tools are currently in development that one can use to check the constraints illustrated above mechanically, which supports the approach by saving time and preventing errors when analyzing the model for critical systems design flaws.

More information can be found in the forthcoming book [Jür04] and articles including [Jür02, Jür03].

4 Biography

Jan Jürjens is a researcher at TU Munich (Germany). He is the author of a book on Secure Systems Development with UML (Springer-Verlag, to be published in March 2004) and about 30 papers in international refereed journals and conferences, mostly on computer security and safety and software engineering, and has given 4 invited talks at international conferences. He has created and lectured a course on secure systems development at the University of Oxford and about 20 tutorials at international conferences. He is the initiator and current chair of the working group on Formal Methods and Software Engineering for Safety and Security (FoMSESS) within the German Society for Informatics (GI). He is a member of the executive board of the Division of Safety and Security within the GI, of the executive boad of the ”QFA Modellierung” committee of the GI, of the advisory board of the Bavarian Competence Center for Safety and Security, of the working group on e-Security of the Bavarian regional government, and of the IFIP Working Group 1.7 ”Theoretical Foundations of Security Analysis and Design”. He has been leading various security-related projects with industry and has acted as a reviewer for EU research projects.

References

[Jür02] J. Jürjens. UMLsec: Extending UML for secure systems development. In J.-M. Jézéquel, H. Hussmann, and S. Cook, editors, *UML 2002 – The Unified Modeling Language*, volume 2460 of *Lecture Notes in Computer Science*, pages 412–425, Dresden, Sept. 30 – Oct. 4 2002. Springer-Verlag.

[Jür03] J. Jürjens. Developing safety-critical systems with UML. In P. Stevens, editor, *UML 2003 – The Unified Modeling Language*, volume 2863 of *Lecture Notes in Computer Science*, pages 360 – 372, San Francisco, Oct. 20–24 2003. Springer-Verlag. 6th International Conference.

[Jür04] J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, April 2004. To be published.