Critical Systems Development with UML: Overview with Automotive Case-study

Jan Jürjens and Johannes Grünbauer
Software & Systems Engineering, TU Munich, Germany
http://www4.in.tum.de/~{juerjens|gruenbau}

Abstract

We give an overview over the content of the tutorial "Critical Systems Development with UML: Methods and Tools". We explain the methodology proposed in the tutorial at the hand of an example from the Automotive domain.

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, [10]) 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 everyday work using UML.

However, there are some challenges one has to overcome to exploit this opportunity, which include the following:

- Adaptation of UML to critical system application domains.
- Correct use of UML in the application domains.
- Conflict between flexibility and unambiguity in the meaning of a notation.
- Improving tool-support for critical systems development with UML.

The tutorial aims to give background knowledge on using UML for critical systems development and to contribute to overcoming these challenges. We show 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 sketch how one can check whether the constraints associated with the stereotypes are fulfilled in a given specification, if desired by performing a mechanical analysis. This way one can find flaws present in the design, before a system is deployed, or even implemented (late correction of requirements errors costs up to 200 times as much as early correction [1]).

Problems in critical systems development often arise when the conceptual independence of software from the underlying physical layer turns out to be an unfaithful abstraction (for example in settings such as real-time or more generally safety-critical systems, see [11]). Since UML allows the modeller to describe different views on a system, including the physical layer, it seems promising to try to use UML to address these problems by modeling the interdependencies between the system and its physical environment.

The approach can also be used to analyze code for weaknesses using model-based criticality testing. There, test sequences are generated from an abstract system specification to provide confidence in the correctness of an implementation. For critical systems, finding tests likely to detect possible failures or vulnerabilities is particularly difficult, as they usually involve subtle and complex execution scenarios (and sometimes the consideration of domain-specific concepts such as cryptography and random numbers).

2. Overview

The tutorial presents the current academic research and industrial best practice regarding the development of critical systems with UML. As example application domains, we focus on safety- and security-critical systems. We also explain how to generalize the approach to other application domains.

We start by giving an overview of UML (the UML diagrams) and model management (packages, subsystems). We explain the UML extension mechanisms (stereotypes, tags, constraints, profiles).

We present extensions of the Unified Modeling Language (UML) for safe resp. secure systems development, called UMLsafe resp. UMLsec, using UML's standard extension mechanisms. We give the UMLsafe and UMLsec profiles.

We show how to formulate safety and security requirements on a system and safety and security assumptions on the underlying physical layer in UMLsafe and UMLsec. We explain how to use this information for risk analysis and how to evaluate the system specification against the safety and security requirements, by making use of a formal behavioral model for a core of UML. Being able to formulate safety and security concepts in the context of a general-purpose modeling language allows encapsulation of established principles of safety and security engineering to avoid common weaknesses introduced by developers without in-depth training in safety and security issues. The formal foundation of the approach allows the discovery of even non-obvious weaknesses that safety and security experts may not detect without use of formal tools.

We discuss applicability of the approach with examples from various application domains.

The tutorial includes a demo of a prototypical tool for the formal analysis of UML models for critical requirements.

2.1. Outline

The tutorial addresses the following main subtopics:

- UML basics, including extension mechanisms
- Applications of UML to
  - dependable systems
  - security-critical systems
  - embedded systems
  - real-time systems
- Extensions of UML (UMLsec, UMLsafe, ...)
- Using UML as a formal design technique for the development of critical systems.
- Model-based testing
- Case studies.
- Advanced tool-support for critical systems development using UML (including synthesis, analysis, testing, validation, and verification). In particular: Using the standard model interchange formats (XMI) for tool integration and to connect to validation engines.

2.2. Goals and Objectives

The tutorial addresses practitioners and researchers in critical systems development interested in using UML and model-based testing (in particular for dependable, security-critical, or real-time systems).

Basic knowledge of object-oriented or component-oriented software is assumed. No specific knowledge of UML or the various application domains is required.

By the end of the tutorial, the participants will have knowledge on how to use the UML and model-based testing for a methodological approach to critical systems development. They will be able to use this approach when developing or analyzing critical systems, by making use of existing solutions and of sound methods of critical systems development.

The tutorial is a sequel in a series of about 20 tutorials presented at international conferences [3].

3. Walk-through using an Automotive Case-study

For adaption to a particular application domain UML provides three "leightweight" extension mechanisms: stereotypes, tagged values and constraints. New types of modelling elements, which extend the semantic in the existing types in the UML metamodel can be defined via stereotypes. They are represented by using 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.

In this section we take a simple, fictitious case-study describing an automotive toll collection system to demonstrate our ideas. Here we focus on the security-critical requirements. Examples for safety-critical requirements can be found in [4].

3.1. Security Requirements Capture with Use Case Diagrams

To describe typical interactions beween a user and a computer system in requirements elicitation, use cases are commonly used. They may also be used for capturing security requirements.

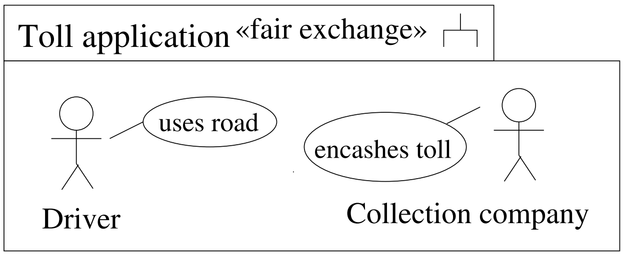

Figure 1. Use case diagram for toll application

To start with our example of a toll collection system: a driver has to pay toll to a company for using a road network. For this purpose, when he gets on the road he will be registered at a toll station. He then passes several toll stations during his trip. When he leaves the road, he has to pay for the stretch of way he drove on. The payment should be performed in a way that prevents either of the involved parties from cheating. For use case diagram in Fig. 1, we include this requirement as the stereotype «fair exchange» attached to the subsystem containing this diagram.

3.2. Secure Business Processes with Activity Diagrams

To model workflow and to explain use cases in more detail, activity diagrams can be used. They are also very helpful to make security requirements more precise.

To continue with our example, Fig. 2 explains the use case in more detail using an activity diagram. We include the *fair exchange* requirement mentioned above . 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 property can actually be checked mechanically).

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.

3.3. Physical Security using Deployment Diagrams

In the UML notation, deployment diagrams are used to describe the physical layer of a system. We use them to check whether the security requirements on the logical level of the system are enforced by the level of physical security, or whether additional security mechanisms (such as encryption) have to be employed.

Following our example, when the driver approaches a toll station, a wireless connection is established. The payment transaction involves transmission of data to be kept

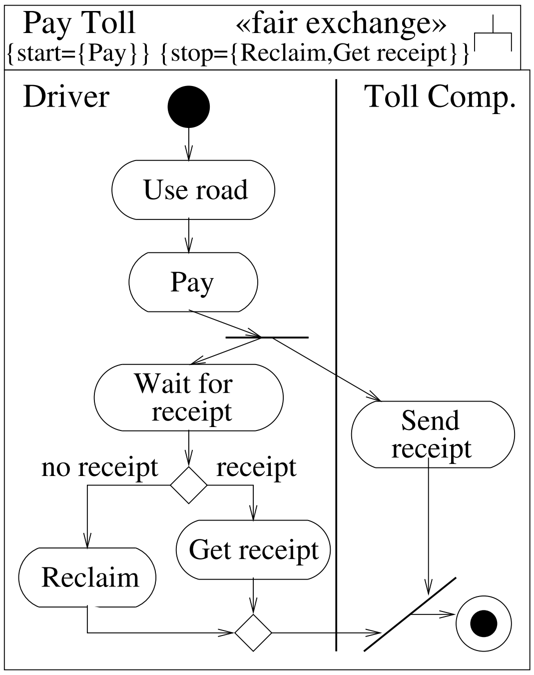

Figure 2. Pay toll activity diagram

secret. Private data should not be visible to an attacker. This information on the physical layer and the security requirement is reflected in the UML model in Fig. 3. We therefore use the stereotype «secure links» to express the constraint that security requirements on the communication are met by the physical layer. More precisely, for each dependency d stereotyped «secrecy» between subsystems or classes on different nodes n, m, and any communication link l between n and m with some stereotype s, the threat scenario arising from the stereotype s with regards to an adversary of a given strength does not violate the secrecy requirement on the communicated data. 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 data that is communicated does not remain secret. For this adversary type, the stereotype is thus applied wrongly to the subsystem. This can also be be checked automatically using appropriate tool-support (see the discussion below).

3.4. Security-critical Interaction with Sequence Diagrams

To specify interaction between different parts of a system, sequence diagrams are used. One can extend them by attaching security requirements to that interaction.

With regards to our example, we specify a core part

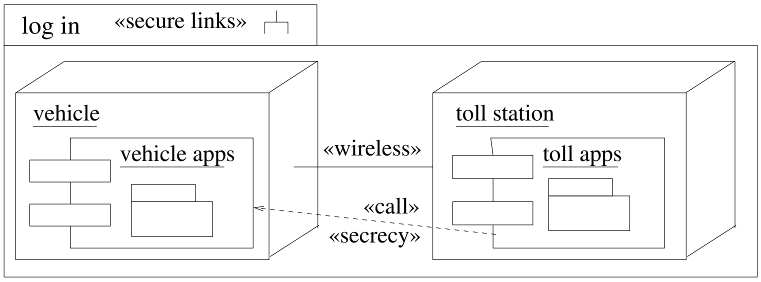

Figure 3. Physical layer of toll application

of a fictitious toll payment protocol using a sequence diagram (see Fig. 4; note that this protocol is used here only to demonstrate our methodology, it is not suggested that this necessarily has to be a useful protocol). Firstly, the driver sends a request to pay at a new toll station together with the data received from the previous toll station he passed to the toll company: his name D, the name TSNo' of the current station, the name TSNo of this last station, the number $\mathsf{counter}$ of stations passed altogether since entering the road, and the point in time when having passed the station. The last three data items had been signed at the previous toll station with the private key of the toll company. All of the data is encrypted with the public key of the toll station. The toll station sends the amount of money the driver has to pay and its own data-set as described above, encrypted with the public key of the driver. The driver acknowledges the data by signing and sending it back to the toll station, again encrypted using the public key of the toll company. Here K_D resp. K_D^{-1} denotes the driver's public resp. private key, K_T and K_T^{-1} the corresponding keys of the toll company.

To include important security requirements on the data that is involved we can again use stereotypes. In this case, the stereotype $\langle\langle\mathsf{critical}\rangle\rangle$ labels classes containing sensitive data and has the associated tags $\{\mathsf{secrecy}\}$ and $\{\mathsf{integrity}\}$ to denote the respective security requirements on the data. To require that these requirements are met relative to the given adversary model, the constraint $\langle\langle\mathsf{data\ security}\rangle\rangle$ is associated with the model. We assume that the standard adversary is not able to break encryption, but is able to exploit any flaw in the protocoll using the "man-in-the-middle"-attack. Technically, the constraint enforces that there are no attacks of that kind. Here we shall point out that it is highly non-trivial to see whether the constraint holds for a given protocol. But using well established concepts from formal methods applied to computer security, it is possible to verify this automatically.

3.5. Secure states using statechart diagrams

Statechart diagrams show the changes in an object's state. They can be used to specify security requirements on the resulting sequences of states and the interaction with the object's environment.

Continuing with the example, the driver is driving along his way, and every time he 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. 5, we can see that a collection company only has access to this list of amounts by using the operation $\mathsf{ga}()$. For privacy reasons we forbid the 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 $\langle\langle\mathsf{no\ down-flow}\rangle\rangle$ 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.

3.6. Tool Support

We describe a prototypical tool [9] currently under development for critical systems development for checking constraints such as those associated with the stereotypes defined above, which is presented at the tutorial. The tool works with UML 1.4 models stored in a XMI 1.2 (XML Metadata Interchange) format by a suitable UML design tool. To avoid having to process UML models directly on the XMI level, the MDR (MetaData Repository,

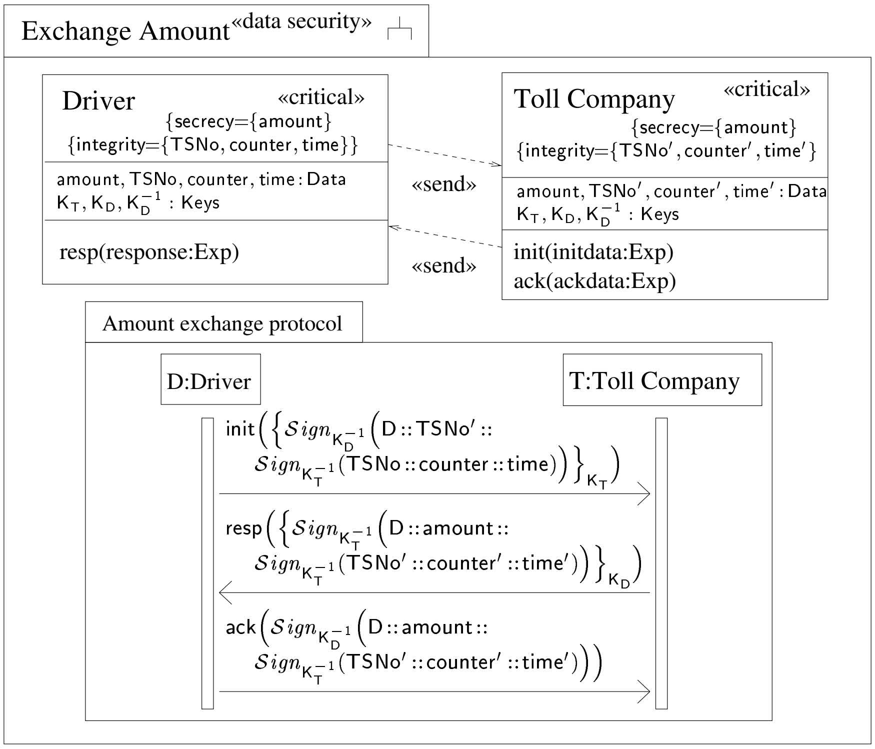

Figure 4. Amount exchange protocol

http://mdr.netbeans.org) is used, allowing one to operate on the UML model level (this is, for example, used by the UML CASE tool Poseidon, http://www.gentleware.com). The MDR library implements a repository for any model described by a modeling language compliant to the MOF (Meta Object Facility). This approach should easen the transition to future UML versions. To use the tool, a developer creates a model and stores it in the UML 1.4 / XMI 1.2 file format. The tool imports the file into the internal MDR repository and 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 problems found, and as a modified UML model, where the stereotypes whose constraints are violated are highlighted. Specifically, the syntactic checks (such as «secure links» and «secure dependency») are implemented in Java, whereas the semantic checks (such as «data security» and «no down − flow») need more specialized tool-support (such as a tool binding with a model-checker currently under development).

4. 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. Alternative approaches can be found for example in [6, 7, 8]. More discussion on related work has to be omitted here for space restrictions, but can be found for example in [5]. Efforts are under way to create a standard UML extension for critical systems development.

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 im-

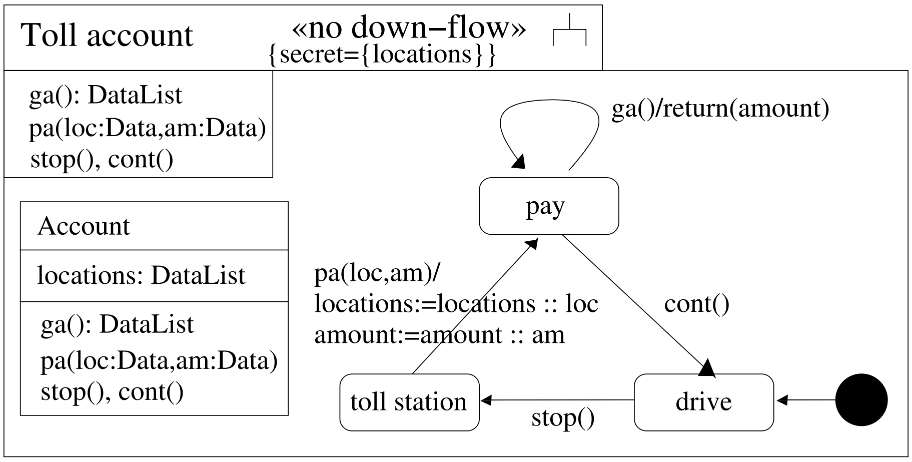

Figure 5. Toll account data object

portant 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 [5], articles including [2, 4], and on the Internet at http://www4.in.tum.de/˜juerjens .

5. References

[1] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[2] 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, Berlin.

[3] J. Jürjens. Critical Systems Development with UML, 2003. Series of tutorials at international conferences. Download of material at http://www4.in.tum.de/˜juerjens/csdumltut .

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

[5] J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, Berlin, 2003. In preparation.

[6] J. Jürjens, V. Cengarle, E. Fernandez, B. Rumpe, and R. Sandner, editors. *Critical Systems Development with UML*, number TUM-I0208 in TUM technical report, 2002. UML'02 satellite workshop proceedings.

[7] J. Jürjens, E. Fernandez, and R. Sandner. Critical systems development with UML-like languages. Special section of the Journal on Software and Systems Modeling, Springer-Verlag, due 2003.

[8] J. Jürjens, B. Rumpe, R. France, and E. Fernandez, editors. *Critical Systems Development with UML*, number TUM-I0323 in TUM technical report, 2003. UML'03 satellite workshop proceedings.

[9] J. Jürjens, P. Shabalin, S. Meng, E. Alter, S. Shen, G. Kokavecz, and S. Schwarzmüller. UMLsec tool, 2003. Webinterface at http://www4.in.tum.de/˜csdumltut.

[10] Object Management Group. OMG Unified Modeling Language Specification v1.5: Revisions and recommendations, Mar. 2003. Version 1.5. OMG Document formal/03-03-01.

[11] B. Selic. Physical programming: Beyond mere logic. In A. Sangiovanni-Vincentelli and J. Sifakis, editors, *Embedded Software Second International Conference (EMSOFT 2002)*, volume 2491 of *Lecture Notes in Computer Science*, pages 399–406, 2002.