Transformations for Introducing Patterns
A Secure Systems Case-study

Jan Jürjens*

Computing Laboratory, University of Oxford, GB

Abstract. We use transformations between UML models to introduce patterns by refinement. The source of the transformations can be parameterised over values for increased flexibility, and the target of the transformation can be a set of models to account for the fact that many patterns can be implemented in several ways. The intended use of our approach is in the context of a formal semantics for UML. We illustrate our approach with examples from secure systems development.

1 Introduction

The usefulness of transformations in computer science has long been recognised. Here we use transformations in the context of the Unified Modeling Language (UML) [RJB99]. More specifically, we use them as refinements for the introduction of patterns within the design process.

We use what we call *generalised transformations* between UML models:

- Firstly, the source of the transformation need not be a single UML model, but it can be *parameterised* over values for increased flexibility.
- Secondly, the target of the transformation (for each parameter) can be a *set* of models to account for the fact that many patterns can be implemented in several ways [KKH00].

The long term aim is to enable mechanical support for introduction of patterns when constructing UML models, and to allow one to check that the patterns are introduced in a way that has previously shown to be useful and correct. Since the envisioned setting is that of a formal semantics for UML (cf. e.g. [EFLR99]), having a sound way of introducing patterns using transformations can ease formal verification (where desired), since the verification can be performed on the more abstract and simpler level. For code generation, one may apply the same principle of introducing patterns using transformations, except that UML models are transformed into code rather than more fine-grained models.

We illustrate our approach with examples from secure systems development, following the approach in [Jür01a]. Having a well-founded way of applying established engineering knowledge is highly desirable in the area of security, since:

- on the one hand, the need to consider security aspects when developing systems is not always met by adequate knowledge on the side of the developer,
- on the other hand, in practice security is compromised most often not by breaking the dedicated mechanisms (such as encryption or security protocols), but by exploiting weaknesses in the way they are being used [And94].

* `http://www.jurjens.de/jan` – `jan@comlab.ox.ac.uk` - Supported by the Studienstiftung des deutschen Volkes and the Computing Laboratory.

Thus security mechanisms cannot be "blindly" inserted into a security-critical system, but the overall system development must take security aspects into account.

Therefore the aim of our work is two-fold:

- exemplarily explain our approach of introducing patterns using generalised transformations (a more complete account has to be given elsewhere) and
- demonstrate the usefulness of introducing patterns in a sound way in the setting of secure systems development.

2 Patterns and UML for Secure Systems Development

Patterns [GHJV95] encapsulate design knowledge of software engineers in the form of recurring design problems. They generally have four core elements: the *pattern name*, the *problem description*, the *solution*, and the *consequences*.

In our approach, a pattern is represented by a multi-valued function (equivalently, a relation) from UML models (giving the problem description) to sets of UML models (the possible solutions). The consequences are given by a set of properties shared by the possible solutions (the resulting set of UML models), which are typically not fulfilled in the UML models giving the problem description. We allow this function to be parameterised by values occuring in the model to allow flexible use of the pattern in similar but different situations.

UMLsec We use an extension of UML [Jür01a] making use of UML's extension mechanisms to specify standard security requirements on security-critical systems. Here we only use statechart diagrams and deployment diagrams. To specify security levels we use the tag {high} to mark model elements such as dependency arrows, links, and data items (attributes, arguments of operations or signals or return values of operations) of the corresponding security level (absence of this tag is interpreted as the level low).[1] Here we only give ideas, the details can be found in [Jür01a].

Statechart diagrams Intuitively, an object *preserves security* if in its statechart diagram S, no low output value depends on {high} input values.

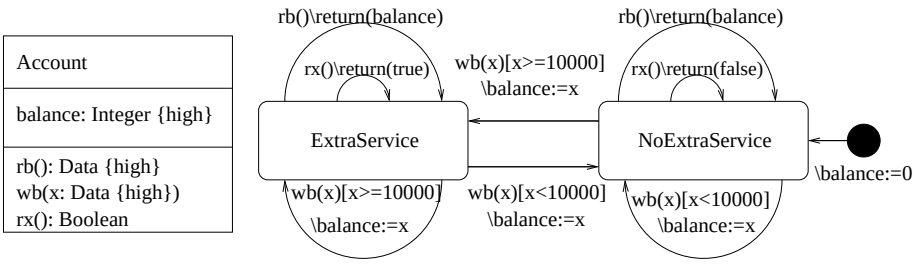


Fig. 1. Multi-level database

Example: Entry in multi-level database The object in Figure 1 does not preserve security (the operation rx() leaks information on the account balance).

[1] More precisely, UML provides tag-value pairs. Here we use the convention that where the values are supposed to be boolean values, they need not be written (then presence of the label denotes the value true, and absence denotes false).

The Wrapper pattern Wrappers [FBF99] are a generic way to augment the security functionality of Commercial Off-The-Shelf (COTS) applications. Here we use wrappers to ensure that objects (that may not be under control of the developer) preserve security as defined above.

Thus the pattern takes any UML object model such as the one in Figure 1 and transforms it into a model by adding a wrapper object that controls its interaction with other objects. One such wrapper is given in Figure 2; it ensures that there can be no low read after a high write [CFMS94]. The way it works is that instead of calling the original object directly, other objects call the wrapper object. This wrapper objects passes the calls on to the object to be wrapped (and gives back the return values to the clients), unless a non-high read method is called *after* a high write method has been called. The *consequence* of the pattern is that the composition of the original object and the wrapper object preserve security.

Our approach can handle other possible solutions since the pattern function is multi-valued. Here the pattern function is parameterised by the number of the methods supplied by the object to be wrapped, and their names.

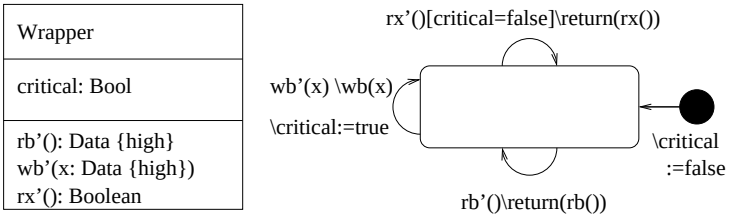
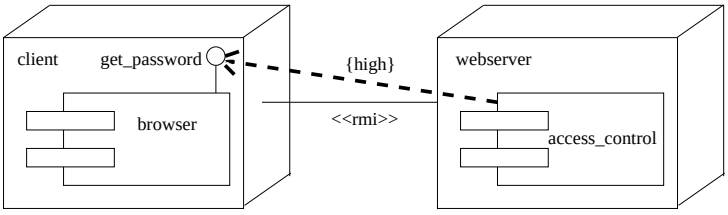


Fig. 2. Wrapper Object

Deployment diagrams A deployment diagram provides *communication security* if for each dependency D that is tagged {high}, the corresponding link L_D is also tagged {high}.

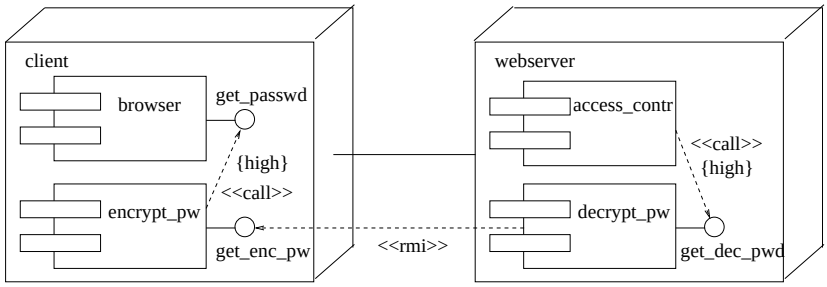
Example



This model does not provide communication security, because the communication link between web-server and client does not provide the needed security level.

The Secure Channel pattern A well-known solution to this problem is to encrypt the traffic over the untrusted link. Here the parameters of the pattern function may include the exact sensitivity level of the data (like "sensitive" or "very sensitive") which translates to different keylengths in the result of the transformation. Also, since the pattern function is multi-valued, we can account for different encryption algorithms (we have to leave out both these aspects). The consequence of this pattern is that the

resulting model provides communication security as defined above since the sensitive data is encrypted before transmission.



3 Related Work

An overview of work on transformations in UML is contained in [Whi00]. [LBE00, KKH00] give results on proving pattern introductions as refinements.

Transformations to enhance, rationalise, refine or abstract UML models are considered in [LB98]. [WS00] gives an algorithm for automatically generating statecharts from sequence diagrams. [KER99] considers refinement of UML diagrams. [GR99, EFG99, ATH00] give transformation rules for class diagrams. Further examples for transformations are given in [WATA00].

There has been much work on security using formal methods (for an overview cf. [RSG+01, AJ01]). Less work has been done using software engineering techniques, in particular we are not aware of any published work that uses UML, besides [Jür01a]. There is considerable work towards providing a formal semantics for UML (for an overview cf. [EFLR99, RACH00, BD00]).

4 Conclusion and Future Work

We used generalised transformations to introduce patterns in UML in the context of a formal semantics. For flexibility, the source of the transformations can be parameterised over values, and the target of the transformation can be a set of models since many patterns can be implemented in several ways. We could only illustrate our approach with examples (from secure systems development), a more complete account will be found elsewhere.

We will consider to what extend transformations preserve desirable properties of a system (in particular security properties, cf. e.g. [Jür01b]). We will develope algorithms for transformations to automatically introduce patterns. We aim to incorporate these into a UML design-tool (such as ArgoUML[2]). For this, we need to extend our approach to incorporate a library of secure systems design patterns.

Acknowledgements This idea for using UML for secure systems development arose when doing security consulting for a project during a research visit with M. Abadi at Bell Labs (Lucent Tech.), Palo Alto, whose hospitality is gratefully acknowledged.

[2] Cf. http://argouml.tigris.org .

References

[AJ01] M. Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. In *Theoretical Aspects of Computer Software (TACS '01)*, LNCS. Springer-Verlag, 2001.

[And94] R. Anderson. Why cryptosystems fail. *Communications of the ACM*, 37(11):32–40, November 1994.

[ATH00] J. Alemán, A. Toval, and J. Hoyos. Rigorously transforming UML class diagrams. In *Workshop MENHIR (Models, Environments, and Tools for Requirements Engineering)*, 2000.

[BD00] C. Bolton and J. Davies. Activity graphs and processes. In *Integrated Formal Methods*, LNCS. Springer-Verlag, 2000.

[CFMS94] S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison Wesley, 1994.

[EFG99] A. Evans, R. France, and E. Grant. Towards formal reasoning with UML models. In *OOPSLA'99 Workshop on Behavioral Semantics*, 1999.

[EFLR99] A. Evans, R. France, K. Lano, and B. Rumpe. The UML as a formal modeling notation. In J. Bezivin and P.-A. Muller, editors, *The Unified Modeling Language - Workshop UML'98: Beyond the Notation*, LNCS. Springer-Verlag, 1999.

[FBF99] T. Fraser, L. Badger, and M. Feldman. Hardening COTS software with generic software wrappers. In *IEEE Symposium on Security and Privacy*, 1999.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GR99] M. Gogolla and M. Richters. Transformation rules for UML class diagrams. In *1st Int. Workshop Unified Modeling Language (UML'98)*, volume 1618 of *LNCS*, 1999.

[Jür01a] J. Jürjens. Towards development of secure systems using UMLsec. In H. Hußmann, editor, *Fundamental Approaches to Software Engineering (FASE/ETAPS, International Conference)*, volume 2029 of *LNCS*, pages 187–200. Springer-Verlag, 2001.

[Jür01b] Jan Jürjens. Secrecy-preserving refinement. In *Formal Methods Europe (International Symposium)*, volume 2021 of *LNCS*, pages 135–152. Springer-Verlag, 2001.

[KER99] S. Kent, A. Evans, and B. Rumpe. UML Semantics FAQ. In A. Moreira and S. Demeyer, editors, *Object-Oriented Technology, ECOOP'99 Workshop Reader*. LNCS 1743, Springer Verlag, 1999.

[KKH00] I. Khriss, R. Keller, and I. Hamid. Pattern-based refinement schemas for design knowledge transfer. *Knowledge-Based Systems*, 13(6):403–415, 2000.

[LB98] K. Lano and J. Bicarregui. Semantics and transformations for UML models. In *UML'98 International Workshop: Beyond the Notation*, 1998.

[LBE00] K. Lano, J. Bicarregui, and A. Evans. Structured axiomatic semantics for UML. In *3rd Workshop on Rigorous Object Oriented Methods*, 2000.

[RACH00] G. Reggio, E. Astesiano, C. Choppy, and H. Hußmann. Analysing UML active classes and associated state machines – A lightweight formal approach. In *FASE2000*, volume 1783 of *LNCS*. Springer-Verlag, 2000.

[RJB99] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[RSG+01] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.

[WATA00] J. Whittle, J. Araújo, A. Toval, and J. Alemán. Rigorously automating transformations of UML behavior models. In *UML 2000 WORKSHOP Dynamic Behaviour in UML Models: Semantic Questions*, 2000.

[Whi00] J. Whittle. Formal approaches to systems analysis using UML: An overview. *Journal of Database Management*, 11(4):4–13, 2000.

[WS00] J. Whittle and J. Schumann. Generating statechart designs from scenarios. In *Proceedings of the 22nd international conference on on Software engineering*, 2000.