

# Rubacon: Automated Support for Model-based Compliance Engineering

Sebastian Höhn  
Institute of Computer Science and Social Studies  
Friedrichstr. 50  
79098 Freiburg, Germany  
hoehn@iig.uni-freiburg.de

Jan Jürjens  
Computing Department  
The Open University  
GB  
<http://www.jurjens.de/jan>

## ABSTRACT

Compliance frameworks, laws and regulations such as Sarbanes Oxley, Basel II, Solvency II, HIPAA etc. demand from companies in a more and more rigorous way to demonstrate that their organisation, processes and supporting IT landscape implement and follow a set of guidelines at differing levels of abstraction. The work presented in this paper aims to contribute to a software engineering process which is driven by security, risk and compliance management considerations.

We concentrate on a part of this approach that focusses on the question how one can use software engineering methods and tools to enforce that the configuration of a system enforces the security policies that arise from business compliance regulations. We present tool support for Model-based Compliance Engineering, i.e. for the model-based development and analysis of software configurations that ensures compliance with security policies. It allows one to check UML models of business applications and their configuration data for adherence to security policies and compliance requirements. The tool is based on standardized data formats, such as UML and XML, which makes its integration into existing business architectures as efficient as possible.

## 1. INTRODUCTION

Compliance frameworks, laws and regulations such as Sarbanes Oxley, Basel II, Solvency II, HIPAA etc. demand from companies in a more and more rigorous way to demonstrate that their organisation, processes and supporting IT landscape implement and follow a set of guidelines at differing levels of abstraction. This immediately impacts the software vendors. They find themselves in a position where they need to demonstrate that their software has been designed and implemented in a way that allows for compliant configuration and operation. Since business processes are run through IT systems, preventive measures and controls required by the regulations can be implemented through IT security mechanisms, including identity management, au-

thentication, role-based access control, auditing, trust mechanisms, etc. Preventing insider and outsider attacks by these means directly contributes to compliance. In essence, this means to specify the properties achieved by the particular decision and to evaluate their potential to implement a control or to mitigate a business risk identified by a regulation.

The work presented in this paper aims to contribute to a software engineering process which is driven by security, risk and compliance management considerations. The process aims to be equipped with tool-support that will enable seamless tracing and evaluation of security requirements, risks and safeguards along the software lifecycle. Moreover, a focus will be put on the transition between risk, security and compliance requirements and the resulting software and architectures. The aim is to identify and develop security architectures and artifacts enabling the realization and enforcement of high-level security requirements.

Here we concentrate on a part of this approach that focusses on the question how one can use software engineering methods and tools to enforce that the configuration of a system enforces the security policies that arise from business compliance regulations. It is a non-trivial task to ensure that the given configuration of an IT system complies with the intended security policy. On the one hand, this arises from the inherent dynamics of permission assignments in business applications, such as temporary delegation of permissions (for example to vacation substitutes) or changes in the business processes. On the other hand, the sheer size of data that has to be analyzed in order to prove the compliance of a configuration with the policy make this task impractical for human users. The motivating application scenario provided by a large German bank contained more than 60,000 entries in the initial user-permission tables.

To this end, the development of tool support for the verification of compliance within complex access control system was the goal of the presented tool. The tool should in particular be applied to analyze SAP security permissions (as motivated by the industrial partner). The current demonstration reports on the design and implementation of this tool. Permissions are given as input in a XML format through an interface from the SAP system. The business application is modeled by a diagram that is edited with standard UML CASE tools and converted to XMI, and our tool checks the permissions against a defined ruleset using analyzers written in Prolog. In order to be easily adaptable to check constraints in other kinds of application software (including non-SAP based applications), the tool has a modular architecture, standardized interfaces, and expressive se-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE'08, May 10–18, 2008, Leipzig, Germany.

Copyright 2008 ACM 978-1-60558-079-1/08/05 ...\$5.00.

curity rules.

The technical foundations on which the tool that is presented here is based, as well as more context for Model-based Compliance Engineering beyond configuration analysis can be found in [7, 8, 9, 5]. In particular, questions such as whether the model used in compliance verification is a faithful reflection of the software as written are beyond the scope of the current paper but are discussed in op.cit., for example making use of model-based testing. The tool itself can be downloaded as open-source from <http://www.telematik.uni-freiburg.de/rubacon>.

## 2. ANALYZING COMPLIANCE WITH SECURITY RULES

Compliance with the overall company security policy has to be checked with respect to a set of security rules which are defined to enforce the security policy.

The correct configuration of secure business applications that have to satisfy company-wide security policies is a challenging task which cannot be accomplished reliably and efficiently in a manual way. Thus there is a necessity for automated tool-support. The tool presented here takes detailed descriptions of the relevant information: the structure of the business application, the business application configuration, and the security and compliance rules written by the security engineer. Using this input data the verification algorithms check whether these rules are obeyed to in the given system, under the given configuration. If the rules cannot be shown to hold this is written to the generated security report. The tool was designed to fulfill the following list of requirements:

- act on configuration of business applications
- automatically generate report of possible violations
- provide flexible report generation modules
- no dependencies with the actual application to be analysed
- efficient, even on large databases
- rule set must be freely configurable

Especially the requirement of independence from an actual business application made a model based approach a viable solution. Because it is inevitable to take into account several aspects of the application, a model allows the easy adjustment to different applications, without additional programming effort. This independence is important even if the tool were designed for a single scenario (for example, the specific bank mentioned in the introduction), since it is inevitable that systems are adapted to new challenges (arising from business goals or technological advances) and hence the tool must be readjusted to the changing system.

According to the conventions published by the Object Management Group (OMG) as the “classical four layer meta-model framework” [15], software systems can be modeled flexibly in an approach based on several layers of information. The tool requires information of the application structure modelled in UML, and configuration data given as XML.

On the one hand *meta-information* is required that describes the data structures of the business application itself. This information is given as a UML model of the relevant aspects of the application. On the other hand, the analyzer needs to know about the configuration data themselves.

As an example for the separation between *model* and *in-*

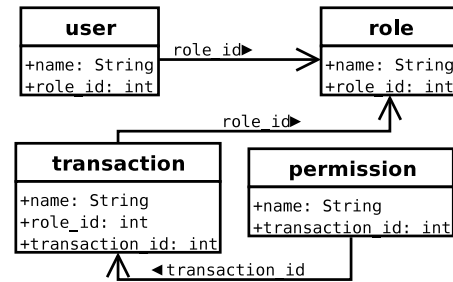


Figure 1: Simple RBAC model

formation consider that there is some user-data in the business application: every user has a name and a role ID. To formally describe the interdependencies of these attributes there exists a UML model that tells the tool about the class `user` and its attributes `name` and `password`. When the tool checks the rules and needs to evaluate information of a given user, e.g. *Alice*, it needs to process the configuration data provided as XML documents.

## 3. SECURITY MODEL

**Application Structure:** The implicit goal of the tool is to investigate chains of information flow, such as determining user permissions from their user names or roles. To bring this information into the context along their actual *resolution path* we provide a static view on the system’s access control system. The UML diagrams used for that purpose are a subset of the class diagrams where the artifacts are represented as classes and the way of deriving information is visualized by directed dependencies. These dependencies are tagged with the connecting attributes (this is required for connecting the structure to the actual configuration data).

This data structure allows one to freely define the static portion of access control systems, such as RBAC or SAP’s hybrid system. To explain this in detail we provide an example of a classical RBAC system (cf. Fig.1. The diagram consists of the classes `user`, `role`, `transaction` and `permission`, with the attributes in Fig. 1 and the associations between these classes with respective connecting elements. This simple model allows the automated derivation of permissions from users’ names.

Note that the direction of these associations is important: when assigning a permission  $p$  to a user  $u$  via a role  $r$ , and the user  $u$  also happens to possess another role  $r'$ , then (of course) it is not admissible to conclude that any user  $u'$  with role  $r'$  should also be granted permission  $p$ . In that sense, assigning roles to users is a *directed* process. In the example diagram, this is expressed using the navigable flag of the UML class diagrams. This flag is an attribute of an association’s end point. If this flag is set to *true* at the end point of

```

<rubacon>
  <user>
    <name>Alice</name>
    <uid>500</uid>
    <role_id>101</role_id>
  </user>
  <role>
    <name>admin</name>
    <role_id>101</role_id>
  </role>
  ...
</rubacon>
  
```

Figure 2: Sample instance file

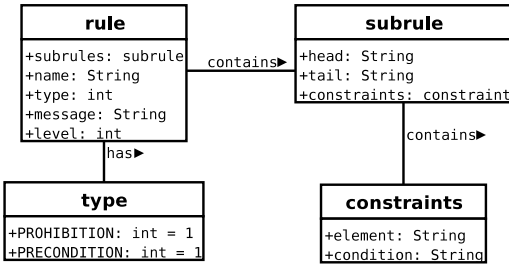


Figure 3: The structure of compliance rules

a class  $c$  (visualized by an arrow at that side of the association), the rule-analyzer may associate information from the other end of the association with  $c$ . If it is set to *false*, this information may not be evaluated. This way the tool may gather permissions with respect to transactions granted to a given user by traversing the class diagram along the associations in the navigable directions by permitting a flow of information. This way the tool collects all users that have a given role, but does not recursively collect all users that have any of the roles that a given user has. Formally this means that the structure represented by the diagram must be a directed graph without circles (since these circles may lead to unintended information collection).

**Configuration Data:** In addition to the structural description of the application explained above we need configuration data. This information is essential for most of the rules that will be of interest in practical applications. Configuration data are always necessary if more than the mere structure of the application is intended to be checked. Configuration data are read by the analyzer as XML files (cf. Fig. 2), containing tags for every class and within these tags one tag for each attribute of the class. The analyzer is capable of generating a XML schema file to verify the input files. This is necessary, because the schema of the input file depends on the subsequent UML model loaded into the tool.

**Security and Compliance Rules:** As already discussed above, the business application data structure is represented by a class diagram, that is, a directed graph together with the data from the business application. These two pieces make up a rather complex graph. For every user in the business application data structure, a node is added in this graph. The model gives the tool the information that there is a connection between a *user* and their *roles*. This is the data structure on which rules are evaluated.

Rules consist of the name, a type, an error message, a priority and a list of sub-rules (cf. Fig. 3). The type is used to define whether the rule describes a precondition (i.e. must hold for every step) or a prohibition (may not occur at all) and the level allows for filtering less important rules from the security report. Sub-rules define a trace in the application graph that the conditions are evaluated at, and the conditions describe the actual operators that are performed on the elements of the sub-trace.

## 4. THE EVALUATION TOOL

For the actual evaluation of the rules on a given configuration in a business application (see Fig. 4 for an example configuration) we use Prolog. Prolog is sufficiently efficient for real-life applications. It is well suited to verify datasets with several million clauses in seconds, while one million entries is a reasonable upper bound for user tables in com-

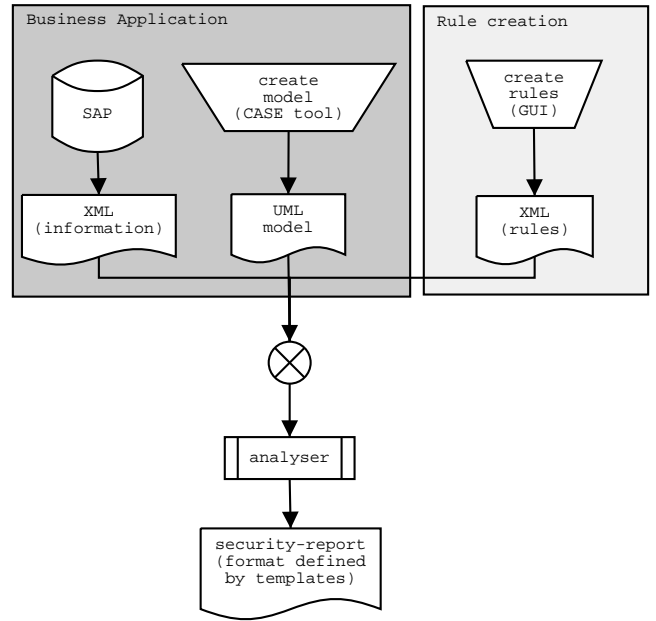


Figure 4: Sample Configuration for Using the Analyser

panies.

As a first step of the transformation the data structure must be translated to Prolog. For each class from the model describing the business application is converted to a predicate with an argument for each attribute. To evaluate expressions like the users' role, we need an additional predicate for each association. The connecting predicates have the following form: if there are predicates  $U(n, r)$  and  $R(m, r)$ , then the association is defined by:  $C(n, m) \rightarrow U(n, r) \wedge R(m, r)$ . These predicates can be extended to paths with any number of intermediate nodes.

To improve performance of the generated code, connecting predicates are only inserted in the program, if they are needed for one of the rules that has to be evaluated. So this reduces the theoretically required  $n \times (n - 1)$  predicates to a maximum of  $m$  predicates to evaluate  $m$  sub-rules. The actual number of predicates is still lower, because not every sub-rule refers to a distinct connecting attributes (e.g. the connection between users and roles is used in more than one sub-rule).

## 5. RELATED WORK

There has been a substantial amount of work on developing tools for analyzing UML models with respect to various requirements, including [3, 11, 14, 2, 6]. Also highly related are XML based checkers such as [17]. However, most of this work is not directly targeted to security requirements. Note that the analysis of security requirements is in general different from that of other kinds of requirements, since it has to take into account the unpredictable adversary behaviour. Therefore it is in general not possible to use general-purpose verification tools for security requirements analysis in a straightforward way (i.e. without hard-coding a non-deterministic adversary model, which makes the verification often computationally very challenging). Our current work is integrated with an approach to analyze security requirements on the basis of UML models that was presented in [8], but it is different from the work presented there in

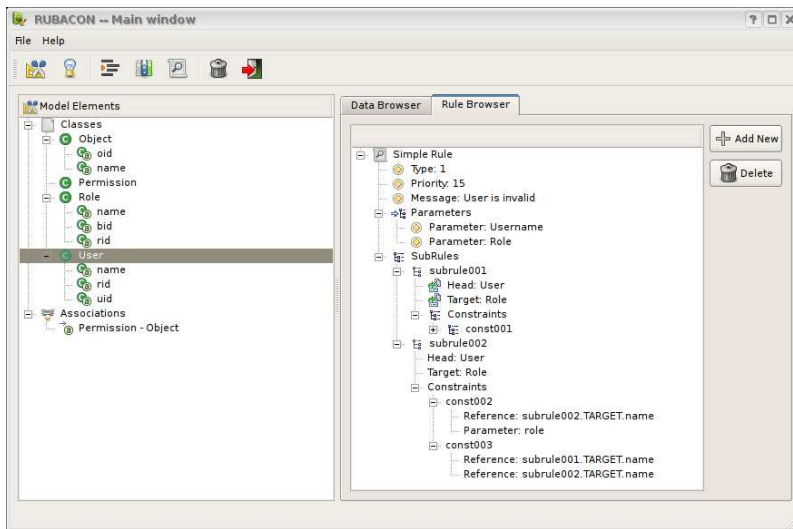


Figure 5: The main window

so far as we are here concerned with analyzing models of security policies and user permissions while [8] deals with cryptographic protocols (which is a very different verification target). The more general context of this work is described in [7] and more on the motivation related to corporate governance and compliance is elaborated in [9]. There are a number of other approaches for secure systems development using UML, including [13, 1, 10, 12]. However, to the extent of our knowledge they do not include tools for verifying user permissions against security policies.

Also related is the work presented in [16] which reports on a formal model for RBAC policy validation and a static-analysis model for RBAC systems that can be used to statically analyze the policy models. Our work differs from that work in that it also applies to other access control models besides RBAC, can also be applied as a run-time analysis (rather than static analysis), which allows one to deal with dynamically changing permissions, and that it is integrated in the general software engineering process by making use of the UML notation.

## 6. CONCLUSION

In this paper we presented tool support for the model-based development and analysis of software configurations that ensures compliance with security policies. It allows one to check UML models of business applications and their configuration data (such as SAP security permissions) for adherence to security policies and compliance requirements. The tool allows the formalization of security rules (such as separation of duty) that the permissions are supposed to satisfy. It enables the verification that the permissions actually implement the rules even in situations where it is difficult, laborious, and error-prone to perform by hand, because of dynamic changes and the size of the data volumes involved.

Experiences with using the tool in practical applications has shown that it can efficiently analyze permission sets in the range found in industrial application (60,000 in the case of the motivating example).

To be easily configurable for a wide variety of business applications (including non-SAP based applications), the analyzer reads the business applications configuration as an UML model and XML files. The rules used for checking are

rather flexible and powerful. While the tool uses a template system for its report the layout of that report can be freely adopted to any form required.

Future work on the tool includes the following: The current graphical user interface is supposed to be extended for easier definition of new rules. A template mechanism for frequent rule patterns will make this process more comfortable which in practice is often preferable to full flexibility [4].

## 7. REFERENCES

- [1] R. Breu and G. Popp. Actor-centric modeling of user rights. In *FASE*, volume 2984 of *LNCS*, pages 165–179. Springer, 2004.
- [2] A. Egyed. Instant consistency checking for the UML. In *ICSE*, pages 381–390, 2006.
- [3] H. Giese, M. Tichy, S. Burmester, and S. Flake. Towards the compositional verification of real-time UML designs. In *ESEC / SIGSOFT FSE*, pages 38–47, 2003.
- [4] S. Höhn. Bringing the user back into control: A new paradigm for usability in highly dynamic systems. In *Trust and Privacy in Digital Business*, LNCS, pages 114–122. Springer, 2006.
- [5] S. Höhn and J. Jürjens. Automated checking of SAP security permissions. In *Integrity and Internal Control in Information Systems VI*, pages 13–30. IFIP, 2004.
- [6] P. Jayaraman and J. Whittle. UCSIM: A tool for simulating use case scenarios. In *ICSE Companion*, pages 43–44, 2007.
- [7] J. Jürjens. *Secure Systems Development with UML*. Springer, 2004.
- [8] J. Jürjens. Sound methods and effective tools for model-based security engineering with UML. In *ICSE*. IEEE, 2005.
- [9] J. Jürjens. Model-based security engineering for compliance with regulatory and business requirements. In *14th Annual Workshop of the HP Software University Association*, 2007.
- [10] D.-K. Kim, I. Ray, R. France, and N. Li. Modeling role-based access control using parameterized UML models. In *FASE*, volume 2984 of *LNCS*, pages 180–193. Springer, 2004.
- [11] S. Konrad, L. Campbell, and B. Cheng. Automated analysis of timing information in UML diagrams. In *ASE*, pages 350–353, 2004.
- [12] Y. Liu and A. Milanova. Ownership and immutability inference for UML-based object access control. In *ICSE*, pages 323–332, 2007.
- [13] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML'02*, volume 2460 of *LNCS*, pages 426–441, 2002.
- [14] I. Ober, S. Graf, and I. Ober. Validating timed UML models by simulation and verification. *STTT*, 8(2):128–145, 2006.
- [15] Object Management Group. *Meta-Object Facility Core Specification Version 2.0*. OMG, January 2006.
- [16] M. Pistoia, S. Fink, R. Flynn, and E. Yahav. When role models have flaws: Static validation of enterprise security policies. In *ICSE*, pages 478–488, 2007.
- [17] xlinkit Rule Workbench, 2005. <http://www.systemwire.com>.