Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 292 (2013) 83–94

www.elsevier.com/locate/entcs

# Towards Security Assurance in Round-Trip Engineering: A Type-Based Approach

Jaime A. Pavlich-Mariscal[1,2]

*Departamento de Ingeniería de Sistemas*
*Pontificia Universidad Javeriana*
*Bogotá, Colombia.*
*Departamento de Ingeniería de Sistemas y Computación*
*Universidad Católica del Norte*
*Antofagasta, Chile.*

María Consuelo Franky[3] Ariel Lopez[4]

*Departamento de Ingeniería de Sistemas*
*Pontificia Universidad Javeriana*
*Bogotá, Colombia.*

Abstract

Security assurance is a property that ensures that the application code behaves consistently with the access control policy specified at the design level. Security assurance proofs are valid as long as software engineers do not modify the generated code. This assumption does not hold in Round-Trip Engineering, since programmers may modify the generated code and the models are automatically re-generated. This paper proposes a round-trip engineering approach for access control that preserves security assurance both when generating code from models and vice versa. The approach is to extend programming languages' typing mechanisms with additional rules that ensure consistency between models and code, even when code is arbitrarily modified by programmers. This paper presents a formal description of the solution and an initial sketch of the required proofs of correctness. Ongoing work is the development of a prototype to automate most of the process and its validation in a case study.

*Keywords:*
Model-Driven Software Engineering, Round-Trip Engineering, Security Assurance, Access Control

## 1 Introduction

Access control is defined as: "Limiting access to information system resources only to authorized users, programs, processes or other systems" [49]. Access control is an

[1] This work is part of the project "Desarrollo de un marco de trabajo para la incorporación de seguridad en software usando Round-Trip Engineering," supported by the Pontificia Universidad Javeriana and Banco Santander S.A.

[2] Email: jpavlich@javeriana.edu.co, jpavlich@ucn.cl

[3] Email: lfranky@javeriana.edu.co

[4] Email: ariel.lopez@javeriana.edu.co


doi:10.1016/j.entcs.2013.02.007

essential component to ensure that sensitive information is secure, uncorrupted, and available. Therefore, it is very important that access control becomes a first-class concern of the software development process. Overall, the general problem that motivates this research is: the need of a process for *secure software engineering* that incorporates access control at every stage in the software development process [39].

Previous work of the authors in this area include a framework for access control modeling and secure code generation [40,38]. At the design level, the framework proposes several extensions to UML to define access control policies based in Role-Based Access Control (RBAC) [41], Mandatory Access Control (MAC) [10], and Discretionary Access Control (DAC) [18]. The focus is to separate access control concerns from other requirements in the design models. At the code level the authors have developed two independent works. One of them is CincoSecurity, a security module based in fine-grained roles and security profiles for Java EE applications [24]. The other work is a set of object-oriented strategies to preserve separation of access control concerns from other requirements [38].

At the code level, a very important issue is *security assurance* [40]. In the context of this research, security assurance means to ensure that the code *correctly* implements the access control from the design. A correct implementation means that the application behaves exactly as the policy intends, allowing subjects to access application resources only if allowed by the rules in the access control design. In support of security assurance, previous work of the authors include a process to formally prove consistency between access control policies and the code that implements them [40]. The above proofs provide security assurance based in the assumption that models are the main development artifact. This premise is valid in some Model-Driven approaches, where designers create an access control policy, then code is automatically generated to implement the policy and the code is not further modified [29].

However, the above assumption does not hold in Round-Trip Engineering (RTE) [29]. In RTE, after the code is generated from models, programmers may modify it. Models can be automatically re-generated from the modified code using a reverse engineering process. This situation may break the consistency between access control models and their implementation in code, since some code configurations may not map to valid access control models.

To address the above problem, this paper proposes an approach to preserve consistency between access control models and code in a round-trip environment. Figure 1 describes the approach. A model of the application is created at the design level, which includes an access control model. Code is automatically generated to implement the design. Programmers can modify the generated code and models can be automatically re-generated using reverse engineering. To ensure that consistency between access control policies and code is not broken, the approach is to extend the typing rules of the programming language. These additional rules restrict the allowed modifications to the code in such a way that, when reverse engineering is performed, only valid access control models are generated.

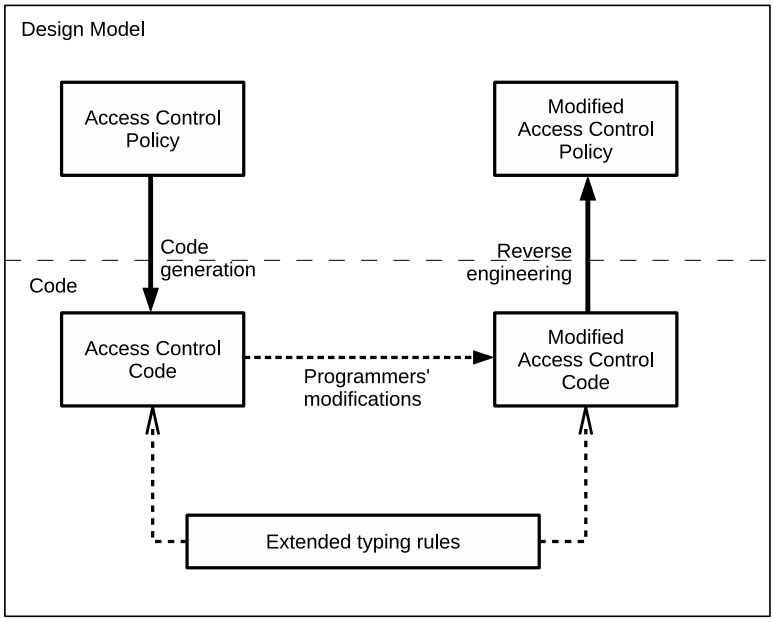


Figure 1. Proposed approach for round-trip security assurance.

The remainder of this paper details the approach. Section 2 explains the essential concepts of the CincoSecurity module [24], which will be used as a case study to illustrate the approach. Section 3 details the proposed approach, formalizing the essential concepts and theorems for security assurance. Section 4 uses a case study based in the CincoSecurity module to develop round-trip security assurance proofs. Section 5 describes some related work. Section 6 concludes this paper.

## 2 CincoSecurity Module

To illustrate the proposed approach, this paper uses the CincoSecurity module as a case study to prove round-trip security assurance. The CincoSecurity module was developed by one of the authors [24] to provide access control enforcement for Java EE applications. CincoSecurity extends the standard access control capabilities of Java EE [47] with Seam [48], providing: enhanced use case implementation and modularization, fine-grained roles, resource protection, security profiles, session and authentication services.

Fine-grained roles will be explained in this section, since this is the main concept utilized in this paper for the case study. Roles usually have associated the privilege to access a specific session bean, and to execute operations over session beans. In CincoSecurity, the role name to access a specific session bean is the same as the name of the bean. Fine-grained roles only have one privilege, to execute a method of a session bean. The role name to access an operation of the session bean has the following format: `"role"_"method"`, where `"role"` is the name of the coarse-grained role to access a session bean and `"method"` is the name of the corresponding operation.

To protect access to resources, CincoSecurity utilizes annotations. To constrain access to session beans, the approach is to add to the bean class an annotation of the form

```
@Restrict("#{s:hasRole('role')}")
```

Where **role** corresponds to the role to access the session bean and it has the same name as the session bean.

To constrain access to operations in the session beans, the approach is to the bean methods, annotations of the form

```
@Restrict("#{s:hasRole('role_operation')}")
```

Where **role** corresponds to the role to access the session bean and **operation** corresponds to the name of the method. When the code is executed in a Java EE application server, the server interprets the above annotations and automatically controls access to the annotated resources. In case an access is restricted, the server throws an exception, which can be managed by programmers to introduce the adequate behavior.

CincoSecurity also adds code to protect JSF pages, but it will not be further detailed, since it is not used in the case study.

## 3 Preserving Consistency between Access Control Policies and Code

In Round-Trip Engineering, designers create models of the system and, using a code generator, automatically create the code that implements those models. In addition, programmers can modify the generated code. Finally, a reverse engineering process parses the modified source code to re-generate the models [29].

In a regular round-trip environment, this process may work without further modifications. For instance, assume that designers model an application structure using a UML class diagram and its behavior with a sequence diagram [34]. Code can be easily generated to implement these diagrams. Further modifications to the code can be reverse-engineered into models using standard CASE tools [29]. Although some information may be lost in the process, such as UML class associations, additional data can be included into the code to preserve that information after the reverse-engineering process [26]. Overall, as long as there is full correspondence between elements in models and code, one can assume that the round-trip process will preserve consistency between both domains.

However, some application concerns introduce challenges to round-trip engineering. Particularly, for access control there may be some code that may break consistency with access control requirements after the reverse engineering process. To better understand this issue, consider a courseware application that includes a `CourseManagement` role with permissions over method `getSyllabus` that retrieves the syllabus of a course. Assume that code is generated to implement the access control model using Java EE with Seam [48]. The generated code will include annotations of the form

```
@Restrict("#{s:hasRole('CourseManagement')}")
```

in all of the methods that are authorized for course management. There are no mechanisms to protect those annotations from changes that programmers could make, e.g., changing role names, defining new roles, etc. A more complex case occurs when using CincoSecurity [24] to implement security. For each method granted to `CourseManagement`, a fine-grained role must be created with a name of the form `"role"_"operation"`. For the above example, the annotation in the `getSyllabus` method would be of the form

```
@Restrict("#{s:hasRole('CourseManagement_getSyllabus')}").
```

In this case, not only the annotations are unprotected from programmers' changes. There is also the risk that the modified code would not yield a valid model when performing reverse engineering. For instance, if programmers change the operation in the annotation to `Course_getSyllabi`, the generated model would be invalid, since the method `getSyllabi` may not exist and the annotation would be defined over a method with a different name (`getSyllabus`). Overall, there is no guarantee that modifications to code would comply with previously defined access control requirements.

To address this problem, the proposed approach is to enhance the typing rules of the target programming language to include automatic checkings for any rule imposed by the access control model and the security library. These typing rules should only allow modifications to the code that yield valid access control models. From these extended typing rules, one can provide security assurance in a round-trip environment, by formally proving that the generated code and models satisfy certain consistency properties.

*Round-Trip Security Assurance*

In a round-trip engineering project, there are several essential elements: models that represent the system, mappings from models to code (code generators), code that implement the models, and reverse engineering mappings (to generate models from code). In addition, models have a set of constraints that assure that the model is valid, which are usually verified by the CASE tool utilized to create them. For instance, a constraint over a UML class model is that there must not be two methods with the same name and parameters in a class.

Similarly, the code has a set of typing rules that ensure that the code is well-formed, and they are usually verified at compile-time. For instance, a typing rule in a Java program is that the types of the variables passed as arguments to a method call must match the types of the parameters of the corresponding method.

For the remainder of this paper, a *round-trip scheme* refers to a collection of four elements: constraints over models, code generation mappings, typing rules, and reverse engineering mappings. The essential idea of round-trip security assurance is to preserve consistency between access control models and code that are being developed using a given round-trip scheme. More formally, round-trip security assurance is achieved through a proof of correctness, where a round-trip scheme is

*correct* if: (a) the generated code correctly implements the access control policy and (b) the code, whether it is modified or not, can yield a valid access control model after the reverse engineering process.

Definitions 3.1 to 3.6 formalize models, code and the components of a round-trip scheme:

**Definition 3.1** $M$ is the set of all possible models of applications, $\mathfrak{I}$ is the set of all possible implementations in a programming language.

**Definition 3.2** Given a set of constraints $K$, a model $m \in M$ is valid over $K$ if $m$ satisfies all of the constraints in $K$.

**Definition 3.3** Given a set of typing rules $T$, an implementation $I \in \mathfrak{I}$ is well-typed if $I$ satisfies all of the typing rules in $T$.

**Definition 3.4** Given a model $m \in M$ of the application and an implementation $I \in \mathfrak{I}$, Code Generation is a function $g : M \to \mathfrak{I}$ that maps models to their corresponding implementations.

**Definition 3.5** Reverse Engineering is a function $re : \mathfrak{I} \to M$ that maps implementations to models.

**Definition 3.6** A Round-Trip Scheme is a tuple $\langle g, re, K, T \rangle$, where $g : M \to \mathfrak{I}$ is a code generation function, $re : \mathfrak{I} \to M$ is a reverse engineering function, $K$ is a set of constraints to determine validity of models, and $T$ is a set of typing rules for implementations.

The above definitions are purposefully abstract. No details about the models and code are given, since they are project-dependent. Using these definitions, Theorem 3.7 formalizes round-trip security assurance as a proof of correctness, which states that a round-trip scheme is correct if the code generated from a valid model is well-typed, and the models generated from a well-typed code are valid.

**Theorem 3.7** *Given a set $K$ of constraints, a set $T$ of typing rules, a set $M_{valid} \subseteq M$ of all valid models under $K$, and a set $\mathfrak{I}_{ok} \subseteq \mathfrak{I}$ of all well-typed implementations over $T$. A round-trip scheme $\langle g, re, K, T \rangle$ is correct if $g(m)$ is well-typed under $T$, for all $m \in M_{valid}$ and $re(I)$ is valid under $K$, for all $I \in \mathfrak{I}_{ok}$.*

# 4 Proof of Correctness for CincoSecurity

To illustrate the use of definitions of Section 3, this section utilizes the CincoSecurity module as a case study, formalizing a simple application model with access control, mappings to and from code that use CincoSecurity, constraints over models, and typing rules over programs. Then it uses all of these definitions to prove security assurance over a given round-trip scheme.

Definitions 4.1 to 4.6 formalize with more details the components of a model (see Definition 3.1). For simplicity, a model in this case study includes only operations (methods) of the application, and the access control policy, which indicates which

operations can access each role. No information about classes are explicitly included, since that would make the definitions and proofs unnecessarily complex for this example.

**Definition 4.1** $R$ is the set of roles that interact with an application.

**Definition 4.2** $Op$ is the set of operations (methods) of an application.

**Definition 4.3** $Obj$ is the set of objects of an application

**Definition 4.4** An Authorization is a tuple $\langle r, op\rangle$, where $r \in R$ is a role that interacts with the application and $op \in Op$ is an operation (method) of the application that the role can execute.

**Definition 4.5** An Access Control Policy $P$ is a set of authorizations.

**Definition 4.6** A Model $m \in M$ of an application is a tuple $\langle Op, P\rangle$, where $Op$ is the set of operations (methods) of the application and $P$ is the access control policy of the application.

Definitions 4.7 to 4.9 describe the essential elements of the code that implement the models. For simplicity, only the essential elements of Java EE are included: operation (method) implementations and annotations. Annotations are represented as a tuple $\langle a, op, v\rangle$, which is equivalent to a Java annotation of the form $@a(v)$ over a method $op$. Annotations may contain any object inside $(v)$, but for the purposes of this case study, we will put strings inside annotations. To adequately interpret the strings within annotations, Definition 4.9 includes an auxiliary function to obtain the string representation of a role or an operation.

**Definition 4.7** An Annotation is a tuple $\langle a, op, v\rangle$, where $a$ is the annotation name, $op$ is the operation being annotated, and $v$ is an object $v \in Obj$.

**Definition 4.8** An Application Implementation $I \in \mathfrak{I}$ is a tuple $\langle IOp, A\rangle$, where $IOp$ is a set of operation implementations, tuples of the form $\langle op, t\rangle$, where $op$ is an operation, and $t$ is its implementation in the programming language. $A$ is a set of annotations over operations in $IOp$.

**Definition 4.9** The string representation of an element, denoted $[e] = s$ is a mapping $[\,] : E \to String$ from a model or implementation element $e \in E$ to its corresponding string representation $s \in String$, $String \subseteq Obj$. In an abuse of notation, $e$ can be either a role or an operation $E = Op \cup R$.

Definition 4.10 specifies a round-trip scheme that uses the CincoSecurity module. The code generation function takes as input a model with operations and a policy, and outputs an implementation with operations and `Restrict` annotations (see Section 2 for more details about these annotations). For simplicity, the text inside `Restrict` annotations only has the essential information of the role and operation. All of the additional syntax is removed to reduce the complexity of the example. Analogously, the reverse engineering mapping takes as input an implementation using CincoSecurity and outputs a model.

Models must satisfy a set $K_{cs}$ of constraints, which ensure that all of the permissions in an access control policy reference existing operations in the model. Similarly, the implementation must satisfy three typing rules. The first two are abstractions of typing rules found in Java: Rule (6) indicates that there must not be duplicate operations in the implementation, Rule (7) indicates that annotations must reference an existing operation (method) in the implementation. Rule (8) represents an extension to the typing rules of Java, to ensure that any implementation using CincoSecurity yields only valid models when performing reverse engineering. This rule indicates that the string of every `Restrict` annotation must include a role name and and operation name, and the operation name must correspond to the operation referenced by the annotation.

**Definition 4.10** The CincoSecurity round-trip scheme is a tuple $\langle g_{cs}, r_{cs}, K_{cs}, T_{cs}\rangle$, where $g_{cs}$ is a code generation function $g_{cs}(\langle Op, P\rangle) = \langle IOp_{cs}, A_{cs}\rangle$, such that

$$(1)\quad IOp_{cs} = \left\{ \begin{array}{c} \langle op, t\rangle \mid op \in Op \\ \wedge \neg\exists \langle op, t'\rangle \in IOp\, s.t\, t \neq t' \end{array} \right\}$$

$$(2)\quad A_{cs} = \{\langle \texttt{Restrict}, op, [r]\_[op]\rangle \mid \langle r, op\rangle \in P\}$$

$re_{cs}$ is a reverse engineering mapping $re_{cs}(\langle IOp, A\rangle) = \langle Op, P\rangle$, such that

$$(3)\quad Op_{cs} = \{op \mid \langle op, t\rangle \in IOp\}$$

$$(4)\quad P_{cs} = \{\langle r, op\rangle \mid \langle \texttt{Restrict}, op, [r]\_[op]\rangle \in A\}$$

$K_{cs}$ is a set of constraints for the validity of models

$$(5)\quad K_{cs} = \{\forall \langle r, op\rangle \in P_{cs}\ op \in Op_{cs}\}$$

and $T_{cs}$ is a set of typing rules $T_{cs} = \{tr_1, tr_2, tr_3\}$, where

$$(6)\quad tr_1 = \forall \langle op, t\rangle \in IOp\ \neg\exists \langle op, t'\rangle \in IOp_{cs},\ t \neq t'$$

$$(7)\quad tr_2 = \forall \langle a, op, v\rangle \in A_{cs}\ \exists \langle op, t\rangle \in IOp_{cs}$$

$$(8)\quad tr_3 = \forall \langle \texttt{Restrict}, op_i, [r]\_[op_j]\rangle \in A_{cs}, op_i = op_j$$

To provide security assurance for the above round-trip scheme, one must prove that it is correct. Theorem 4.13 is the application of Theorem 3.7 to prove correctness of the scheme of Definition 4.10. To facilitate the proof, Lemma 4.11 proves correctness for the code generation mapping $g_{cs}$ and Lemma 4.12 proves correctness of the reverse engineering function $re_{cs}$.

**Lemma 4.11** *For all valid models $m = \langle Op, P\rangle$ under $K_{cs}$, $g_{cs}(m)$ is well-typed under $T_{cs}$.*

**Proof** To prove that $g_{cs}(m)$ is well-typed under $T_{cs}$, one must first prove that $g_{cs}(m)$ satisfies each of the constraints in $T_{cs}$.

- From (1), $IOp_{cs}$ is the set of tuples of the form $\langle op, t\rangle$, where there are no two tuples having the same $op$. Therefore, Rule (6) of $T_{cs}$ is satisfied.

- From (2), $A_{cs}$ is a set of tuples of the form $\langle \texttt{Restrict}, op, [r]\_[op] \rangle$, where $\langle r, op \rangle$ belongs to the policy $P$ of $m$. Since $m$ satisfies the constraint of $K_{cs}$ (5), all of the operations $op$ in $P$ also belong to $Op$. Therefore, for all $op$ in the generated annotations $\langle \texttt{Restrict}, op, [r]\_[op] \rangle$, $op \in IOp_{cs}$, which satisfies Rule (7) of $T_{cs}$.
- From (6), (5), and (2), it follows that Rule (8) is also satisfied. □

**Lemma 4.12** *For all well-typed implementations $I \in \mathfrak{I}$ under $T_{cs}$, $re_{cs}(I)$ is a valid model under $K_{cs}$.*

**Proof** To prove the validity of $re_{cs}(I)$, one must prove that $re_{cs}(I)$ satisfies the constraints in $K_{cs}$ (5). From (3), there is an operation $op \in Op_{cs}$ for each $\langle op, t \rangle \in IOp$. Since $I$ satisfies (7) and (8), and from (4), it follows that all operations $op$ in $P_{cs}$ also belong to $Op_{cs}$, which satisfies the constraint in $K_{cs}$. □

**Theorem 4.13** *The CincoSecurity round-trip scheme $\langle ge_{cs}, r_{cs}, K_{cs}, T_{cs} \rangle$ is correct.*

**Proof** Straightforward from Theorem 3.7, and Lemmas 4.11 and 4.12. □

## 5 Related Work

There are several works that incorporate access control into software. At the design level, UMLSec [28], AuthUML[6], Doan et al. [17] propose extensions to UML to model different aspects of RBAC, MAC, or both. Song et al. [45] and Mouheb et al. [32] define aspect-oriented mechanisms to transform software models to incorporate access control requirements. None of these works solve the problem of secure code generation from model specifications.

At the code level, there are also several works. Farias [21], Evans et al.[20], Pandey et al. [37], and Erlingsson et al. [19] provide the compilation process in diverse platforms to incorporate security precondition checking. Similarly, Mourad et al. [33], Alhadidi et al. [7], Bodkin [11], Dantas [14], De Win [15], Huang et al. [27], Shlowikowski et al. [44], Sewe [43] and Viega et al. [50], use aspect-oriented programming to restrict access based in permissions. Frameworks, such as JPA Security[2], FleXive [1], and Seam [48] provide access control mechanisms in Java EE. Centonze et al. [12] and Fischer et al. [22] provide approaches to verify consistency of access control policies in Java EE. None of these approaches provide a traceable link from design models in RBAC, MAC, or DAC to code or vice versa.

Most of the above work focus either in models or code, but not both. The only exception is the work of Basin et al. [9] that provides mechanisms to translate models into code and configuration files. However, their work does not provide any reverse engineering mechanism.

The problem of consistency between models and code can be generalized to the problem of bidirectional transformations [13]. The Query-View Transformation language (QVT) [36] provides support for bidirectional transformations, however, as Stevens et al point out [46], there are several issues in the QVT specification that limits its applicability for model synchronization. Diskin et al [16] provide a formal algebraic specification for bidirectional transformations, with a focus in

model synchronization. Diskin's work, although it is mainly theoretical, it provides some useful artifacts for Round-Trip Engineering. A similar work is presented by Foster et al [23], with a focus in bidirectional tree transformation. Hermann et al [25] proposes an approach for model synchronization based in triple-graph grammars (TGG) [42]. Similarly, Anjorin et al [8] synchronizes models and code using triple-graph grammars, and maps grammars to meta-models. Anjorin et al approach uses TGG at the model level and the meta-model TGG is automatically derived from the TGG of the model-level. However, this relies on the assumption that the abstract syntax tree meta-model is fixed [8]. This assumption is not valid in an MDE environment, where it is necessary to generate code for multiple languages and platforms [30,31,35].

Finally, there are several tools that support code generation and reverse engineering for general-purpose models [3,4,5]. To the best of our knowledge, there are no tools that explicitly address access control requirements in Round-Trip Engineering.

## 6 Conclusions and Future Work

This paper proposed a formal proof of correctness for access control models and code in a round-trip environment. Its applicability was illustrated in a case study using the CincoSecurity module. From the case study one can infer that the proposed approach is applicable for code generation and reverse engineering mappings, provided that models and code are sufficiently abstracted.

As such, this is a first step towards a full-featured round-trip engineering mechanism for access control. Ongoing work is to develop a proof-of-concept prototype that implements all of the proposed formalisms, to automatically check the extended typing rules of the access control code. Future work includes automatically deriving the extended typing rules from the constraints in the model and the code generation mappings and verifying correctness. This will require to explore proof assistants and constraint checking tools, which may provide the basis for such application.

The future results of this work are expected to benefit code generation and reverse engineering by improving safety in the definition of round-trip engineering schemes. A reduction in the loss of information in the mappings is expected, to comply with the correctness conditions. Moreover, since the extended typing rules are associated to code, they can be incorporated into standard IDE tools. Automatic typing rule verifications in these tools are expected to reduce the probability of errors, thus improving the overall round-trip process.

## References

[1] [fleXive] - the Next-Generation open source content repository | www.flexive.org. http://www.flexive.org/.

[2] JPA security. http://jpasecurity.sourceforge.net/.

[3] Borland together. http://www.borland.com/us/products/together/, 2010.

[4] Gentleware - model to business: gentleware homepage. http://www.gentleware.com/,2010.

[5] IBM developerWorks : Rational rose. http://www.ibm.com/developerworks/rational/products/rose/, 2010.

[6] K. Alghathbar and D. Wijesekera. AuthUML: a three-phased framework to analyze access control specifications in use cases. In *Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering*, 2003.

[7] D. Alhadidi, N. Belblidia, M. Debbabi, and P. Bhattacharya. \\lambda\\ _saop: a security AOP calculus. *The Computer Journal*, 2009.

[8] A. Anjorin, M. P Lauder, M. Schlereth, and A. Schürr. Support for bidirectional Model-to-Text transformations. *Electronic Communications of the EASST*, 36, 2011.

[9] David Basin, Jurgen Doser, and Torsten Lodderstedt. Model driven security: from UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15(1):39–91, January 2006.

[10] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations model. Technical report, Mitre Corporation, 1975.

[11] R. Bodkin. Enterprise security aspects. In *Proceedings of the AOSD Technology for Application-level Security Workshop*, volume 9, 2004.

[12] Paolina Centonze, Gleb Naumovich, Stephen J Fink, and Marco Pistoia. Role-Based access control consistency validation. In *Proceedings of the 2006 international symposium on Software testing and analysis*, ISSTA '06, page 121–132, New York, NY, USA, 2006. ACM. ACM ID: 1146253.

[13] Krzysztof Czarnecki, J. Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James Terwilliger. Bidirectional transformations: A Cross-Discipline perspective. In Richard Paige, editor, *Theory and Practice of Model Transformations*, volume 5563 of *Lecture Notes in Computer Science*, pages 260–283. Springer Berlin / Heidelberg, 2009.

[14] Dantas. *Analyzing security advice in functional aspect-oriented programming languages*. PhD thesis, Princeton University, 2007.

[15] B. De-Win. *Engineering application-level security through aspect-oriented software development*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2004.

[16] Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki, Hartmut Ehrig, Frank Hermann, and Fernando Orejas. From state- to Delta-Based bidirectional model transformations: The symmetric case. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems*, volume 6981 of *Lecture Notes in Computer Science*, pages 304–318. Springer Berlin / Heidelberg, 2011.

[17] Thuong Doan. *A Framework for Software Security in UML with Assurance*. PhD thesis, The University of Connecticut, 2008.

[18] DoD. *Trusted Computer System Evaluation Criteria. 5200.28-STD*. DoD, 1985.

[19] Erlingsson and Schneider. SASI enforcement of security policies: A retrospective. In *WNSP: New Security Paradigms Workshop*. ACM Press, 2000.

[20] David Evans and Andrew Twyman. Flexible Policy-Directed code safety. *Security and Privacy, IEEE Symposium on*, page 0032, 1999.

[21] A. Farias. *Towards a Security Aspect for Java*. PhD thesis, Vrije Universiteit Brussel, 2001.

[22] Jeffrey Fischer, Daniel Marino, Rupak Majumdar, and Todd Millstein. Fine-Grained access control with Object-Sensitive roles. In *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, Genoa, page 173–194, Berlin, Heidelberg, 2009. Springer-Verlag. ACM ID: 1615197.

[23] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. *SIGPLAN Not.*, 40(1):233–246, January 2005.

[24] María Consuelo Franky, Toro C, and Victor Manuel. CincoSecurity: automating the security of java EE applications with Fine-Grained roles and security profiles. *International Journal On Advances in Security*, 4(3 and 4):245–254, April 2012.

[25] Frank Hermann, Hartmut Ehrig, Fernando Orejas, Krzysztof Czarnecki, Zinovy Diskin, and Yingfei Xiong. Correctness of model synchronization based on triple graph grammars. In Jon Whittle, Tony Clark, and Thomas Kühne, editors, *Model Driven Engineering Languages and Systems*, volume 6981 of *Lecture Notes in Computer Science*, pages 668–682. Springer Berlin / Heidelberg, 2011.

[26] Hibernate. *Hibernate Persistence Library*. 2012. http://www.hibernate.org/.

[27] M. Huang, C. Wang, and L. Zhang. Toward a reusable and generic security aspect library. In *AOSD: AOSDSEC*, volume 4, 2004.

[28] J. Jurjens. UMLsec: extending UML for secure systems development. In *Proceedings of the 5th International Conference on The Unified Modeling Language*, 2002.

[29] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, 1 edition, March 2008.

[30] Stuart Kent. Model driven engineering. volume 2335 of *Lecture Notes in Computer Science*, pages 286–298. Springer Berlin / Heidelberg, 2002.

[31] Stephen J. MELLOR, Kendall Scott, Axel Uhl, and Dirk Weise. *MDA Distilled*. Addison-Wesley Professional, March 2004.

[32] Djedjiga Mouheb, Chamseddine Talhi, Vitor Lima, Mourad Debbabi, Lingyu Wang, and Makan Pourzandi. Weaving security aspects into UML 2.0 design models. In *Proceedings of the 13th workshop on Aspect-oriented modeling*, pages 7–12, Charlottesville, Virginia, USA, 2009. ACM.

[33] Azzam Mourad, Marc-André Laverdière, and Mourad Debbabi. A high-level aspect-oriented-based framework for software security hardening. *Information Security Journal: A Global Perspective*, 17(2):56, 2008.

[34] OMG. UML 2.0 superstructure. Technical report, Object Management Group, 2005.

[35] OMG. Model driven architecture (MDA). http://www.omg.org/mda/, 2009.

[36] QVT OMG. Meta object facility (MOF) 2.0 Query/View/Transformation specification. Technical report, 2008.

[37] R. Pandey and B. Hashii. Providing Fine-Grained access control for mobile programs through binary editing. Technical Report TR-98-08, 1998.

[38] J. A. Pavlich-Mariscal, Steven A. Demurjian, and Laurent D. Michel. A framework of composable access control features: Preserving separation of access control concerns from models to code. *Computers & Security*, 29(3):350–379, May 2010.

[39] J.A. Pavlich-Mariscal. *A Framework of Composable Security Features: Preserving Separation of Security Concerns from Models to Code*. PhD thesis, University of Connecticut, 2008.

[40] J.A. Pavlich-Mariscal, Steven A. Demurjian, and Laurent D. Michel. A framework for security assurance of access control enforcement code. *Computers & Security*, In Press, Accepted Manuscript, 2010.

[41] R. Sandhu, D. Ferraiolo, and R. Kuhn. The NIST model for Role-Based access control: Towards a unified standard. In *Proceedings of the Fifth ACM Workshop on Role-Based Access Control*, 2000.

[42] Andy Schürr and Felix Klar. 15 years of triple graph grammars. In Hartmut Ehrig, Reiko Heckel, Grzegorz Rozenberg, and Gabriele Taentzer, editors, *Graph Transformations*, volume 5214 of *Lecture Notes in Computer Science*, pages 411–425. Springer Berlin / Heidelberg, 2008.

[43] Sewe, Bockisch, and Mezini. Aspects and class-based security: a survey of interactions between advice weaving and the java 2 security model. In *VMIL '08: Proceedings of the 2nd Workshop on Virtual Machines and Intermediate Languages for emerging modularization mechanisms*, page 1–7, New York, NY, USA, 2008. ACM.

[44] P. Shlowikowski and K. Zielinski. Comparison study of aspect-oriented and container managed security. In *AAOS2003: Analysis of Aspect Oriented Software. Workshop held in conjunction with ECOOP*, 2003.

[45] E. Song. *An aspect-based approach to modeling access control policies*. PhD thesis, Colorado State University, 2007.

[46] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. In Gregor Engels, Bill Opdyke, Douglas Schmidt, and Frank Weil, editors, *Model Driven Engineering Languages and Systems*, volume 4735 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg, 2007.

[47] SUN. Java EE. http://java.sun.com/javaee/, 2010.

[48] SEAM JBoss Team. Seam framework - API and reference documentation. http://seamframework.org/Seam3/APIAndReferenceDocumentation.

[49] ATIS Telecom. *Glossary 2000. T1.523-2001.* 2001.

[50] J. Viega, J. T. Bloch, and P. Chandra. Applying Aspect-Oriented programming to security. *Cutter IT Journal*, 2001.