Tool support for code generation from a UMLsec property

Conference or Workshop Item

How to cite:


For guidance on citations see FAQs.

© 2010 The Authors

Version: Accepted Manuscript

Link(s) to article on publisher’s website:
http://soft.vub.ac.be/ase2010/

Copyright and Moral Rights for the articles on this site are retained by the individual authors and/or other copyright owners. For more information on Open Research Online’s data policy on reuse of materials please consult the policies page.

oro.open.ac.uk
Tool Support for Code Generation from a UMLsec Property*

Lionel Montrieux†
The Open University
Milton Keynes
United Kingdom
L.M.C.Montrieux@open.ac.uk

Jan Jürjens
TU Dortmund & Fraunhofer
ISST
Dortmund
Germany
http://jan.jurjens.de

Charles B. Haley
The Open University
Milton Keynes
United Kingdom

Yijun Yu
The Open University
Milton Keynes
United Kingdom
Y.Yu@open.ac.uk

Pierre-Yves Schobbens
University of Namur
Namur
Belgium
pyshoppen@yahoo.com

Hubert Toussaint
University of Namur
Namur
Belgium
hto@info.fundp.ac.be

ABSTRACT
This demo presents a tool to generate code from verified Role-Based Access Control properties defined using UMLsec. It can either generate Java code, or generate Java code for the UML model and AspectJ code for enforcing said RBAC properties. Both approaches use the Java Authentication and Authorization Service (JAAS) to enforce access control.

Categories and Subject Descriptors: D.2.2 Software Engineering: Design Tools and Techniques [Computer-aided software engineering (CASE)]

General Terms: Security

1. INTRODUCTION
Security requirements can be made explicit on the design level, such as annotations on a UML model. UMLsec [4] extends UML to allow one to express security properties on a model, but it is still the developer’s responsibility to implement the code that will actually enforce those properties. This process can generate bugs and will not give any guarantee about how the implementation conforms to the model.

In this demo, we present a tool that generates Java and AspectJ code from a UML with a verified UMLsec property. It can either generate only Java code, or, alternatively, implement the security property using AspectJ while still using Java for the functional code. The tool also has other features for UMLsec models verification that are not discussed here.

The next sections are organised as follows: we first give a short overview of UMLsec in section 2, then in section 3 we describe the tool, with a particular attention towards the new features we are focusing on in this demo. In section 4 we discuss related work, and we finally discuss future works in section 5.

2. EXPRESSING ACCESS CONTROL AS AN UMLsec PROPERTY
UMLsec [4] is an UML profile allowing one to define security properties, using standard UML extension mechanisms like stereotypes and tagged values. One of those properties that can be defined on a UML model is Role-Based Access Control. A UML activity diagram can be annotated to assign roles to users, grant permissions to roles, and protect actions. Each swimlane in the activity diagram represents a user. It is therefore possible to check the defined RBAC property by making sure no protected action is in the swimlane of a user that is not allowed to perform it. Currently, only a subset of the RBAC standard is supported by the UMLsec specification. For example, it is assumed that all roles are granted to a user at the start of a session, and that no roles can be dropped or delegated to another user.

3. ENFORCING ACCESS CONTROL PROPERTIES THROUGH CODE GENERATION
The UMLsec tool [2] allows one to check whether or not a model enforces a UMLsec property [5]. It also allows one to generate code conforming to the model.
This paper presents two approaches to generating Java code to enforce RBAC properties on a UML model. The first approach produces only Java code, while the second one produces Java code for implementing the RBAC property as an aspect. Both approaches have been implemented using AspectJ. The first approach has the advantage of not requiring the use of AspectJ, but it leads to a more complex code generation process. The second approach is simpler to implement, but it requires the use of AspectJ, which can be more difficult to integrate into existing code bases.

The first approach is based on the idea of adding new lines of code at specific points in the original code. This is done by defining a pointcut that specifies the location of the code to be added, and a Advice that defines the new code. For example, to protect a method called `myMethod` with the RBAC property, the code

```java
public void myMethod() {
    new MyClassPermission("myMethod");
}
```

is added to the original code. This approach is similar to the one used in the previous section, where new code was added to protect a method.

The second approach is based on the idea of using AspectJ to generate code that enforces RBAC properties. This is done by defining an aspect that defines the RBAC property and then weaving it into the original code. For example, to protect a method called `myMethod` with the RBAC property, the code

```java
@MyClassPermission(myMethod)
public void myMethod() {
    MyClass myClass = new MyClass();
    myClass.myMethod();
}
```

is added to the original code. This approach is similar to the one used in the previous section, where new code was added to protect a method.

Both approaches have their advantages and disadvantages. The first approach is simpler to implement, but it leads to a more complex code generation process. The second approach is simpler to implement, but it requires the use of AspectJ, which can be more difficult to integrate into existing code bases.

In future work, we will add support for code generation using other security frameworks than JAAS, like EJB. We will also work on code generation from other UMLsec properties. This will raise new and interesting challenges, as we will need to generate code that enforces several different properties, without introducing conflicts.

6. REFERENCES


FIGURE 3: Sample code added to protect a method called `myMethod` to an access control aspect.