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.
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
pyenschobbens@fundp.ac.be

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 demo focuses on a new feature: code generation from a UML model with an UMLsec RBAC property, as described on Figure 1. Given an UML model with UMLsec annotations describing an RBAC property, we generate code that conforms to the RBAC property. Two different approaches have been implemented, both using the JAAS [1] framework: the first one produces only Java code, while the second one uses AspectJ to enforce the RBAC property.

Both code generation techniques have been implemented using Aspect-Oriented Programming [3]. While code generation from the UML model is done in Java, we use AspectJ to add generation of code enforcing the UMLsec property, which has several advantages over an Object-Oriented only implementation. First, it allows us to clearly separate the code generation of the UML model itself from the code generation of the associated UMLsec property. Second, it allows us to easily extend the tool to generate code from other UMLsec properties by simply writing a new aspect, and reusing the existing Java code for generating to code corresponding to the UML model. And finally, it allows us to study the potential conflicts between several UMLsec properties in terms of an aspect composition problem.

### 3.1 Object-Oriented code generation

The first approach produces only Java code, both for implementing the UML model and the UMLsec property. The aspect responsible for generating code enforcing the RBAC property simply monitors the code generation process, and adds the necessary lines of code in the places where they are needed. The access control code is therefore spread all over the code base. While it might make modifications like adding (resp. removing) access control protection to a non-protected (resp. from a protected) method or attribute, it has the advantage of not requiring the use of Aspect-Oriented Programming and its drawbacks, like potential conflicts between aspects or negative impact on performances. Figure 2 shows an example of the lines of code added to protect a method.

<table>
<thead>
<tr>
<th>public void myMethod() {</th>
<th>public void MyClass.myMethod() {</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessController.checkPermission(new</td>
<td></td>
</tr>
<tr>
<td>MyClassPermission(&quot;myMethod&quot;)));</td>
<td></td>
</tr>
</tbody>
</table>
| Subject.doAsPrivileged(authenticatedSubject,
|    new PrivilegedAction() { |
|    public Object run() { |
|    MyClass.myMethod(); |
|    return null}; }, null); |

Figure 2: Sample code added to protect a method and to call it

This code snippet shows an example of the lines of code added to protect a method.

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

### 3.2 Aspect-Oriented code generation

The second approach produces Java code for implementing the UML model, but the UMLsec property is implemented using AspectJ. Here, the RBAC code generation aspect monitors the code generation and simply creates an aspect while the functional code is being generated. This keeps the functional code separated from the authorisation code, making the modification of the code easier, and helping maintaining a traceability link between the access control code and the UMLsec property. Furthermore, any subse-

```java
Subject.doAsPrivileged(authenticatedSubject, new PrivilegedAction() {
    public Object run() {
        MyClass.myMethod();
        return null;
    }
});
```