SSG: A Model-Based Development Environment for Smart, Security-Aware GUIs

Miguel A. García de Dios
IMDEA Software Institute
miguel.garcia@imdea.org

Carolina Dania
IMDEA Software Institute
carolina.dania@imdea.org

Michael Schläpfer
ETH Zürich
michschl@inf.ethz.ch

David Basin
ETH Zürich
basin@inf.ethz.ch

Manuel Clavel
IMDEA Software Institute
manuel.clavel@imdea.org

Marina Egea
ETH Zürich
marinae@inf.ethz.ch

ABSTRACT

We present a development environment for automatically building smart, security-aware GUIs following a model-based approach. Our environment consists of a number of plugins that have been developed using the Eclipse framework and includes three model editors, a model-transformation tool, and a code generator.

1. INTRODUCTION

In many programs, users access application data using GUI widgets: data is created, deleted, read, and updated using text boxes, check boxes, buttons, and the like. There is an important, but little explored, link between visualization and security. When the application data is protected by an access control policy, the application GUI should be *aware of* and *respect* this policy. For example, the GUI should not display options to users for actions that they are not authorized to execute on application data. This prevents user frustration, for example, from filling out a long electronic form only to have the server reject it because the user lacks a permission to execute some associated action on the application data. Taking this idea one step further, the GUI should not, for example, display options to users to open other widgets when these widgets only display options for actions that the users are not authorized to execute on application data. That is, the application GUI should not just be security-aware but also *smart*.

Here we present an environment for developing such GUIs, using the Eclipse framework. Our environment supports linking visualization and security during system design and using this design information to automatically generate GUI implementations that are both smart and security-aware. To the best of our knowledge, no other GUI development environment currently provides this kind of support, either for design or implementation.

ICSE '10 Cape Town, South Africa

2. MODEL-BASED DEVELOPMENT OF SMART, SECURITY-AWARE GUIS

The default, ad-hoc solution to the problem of linking visualization and security would be to directly hardcode the security policy within the GUI. But this is clearly inadequate. First, the GUI designer is often not aware of the application data security policy. Second, even if the designer is aware of it, hardcoding the application data security policy within the GUI code is cumbersome and error-prone, if done manually. Finally, any changes in the security policy will require manual changes to the GUI code where this policy is hardcoded, which again is a cumbersome and error-prone task.

In [1] we propose a model-based approach to linking visualization and security. The key idea is that this link is ultimately defined in terms of *data actions*, since data actions are both controlled by the security policy and triggered by the events supported by the graphical user interface. The key component of this solution is a many-models-to-model transformation which, given a security-design model and a GUI model, automatically generates a GUI model that is both security-aware and smart.

Thus, under this model-based development approach, illustrated in Figure 1, the process of building a smart, security-aware GUI has the following parts.

1. Software engineers specify both the application-data model $\mathcal{C}$ and the security-design model $\mathcal{S}_\mathcal{C}$.
2. GUI designers specify the application GUI model $\mathcal{G}_\mathcal{C}$.
3. A many-models-to-model transformation automatically generates a smart, security-aware GUI model $\mathcal{M}_{(\mathcal{G}_\mathcal{C},\mathcal{S}_\mathcal{C})}$ from the security-design model $\mathcal{S}_\mathcal{C}$ and the GUI model $\mathcal{G}_\mathcal{C}$.
4. A code generator automatically produces the smart, security-aware GUI from the smart, security-aware GUI model $\mathcal{M}_{(\mathcal{G}_\mathcal{C},\mathcal{S}_\mathcal{C})}$.

As a design methodology, our model-based approach has three main advantages over traditional approaches to user interface design. First, security engineers and GUI designers can independently model what they know best. Second, security engineers and GUI designers can independently change their models, and these changes are automatically propagated to the security-aware GUI models. Third, GUI designers can use the generated security-aware GUI models

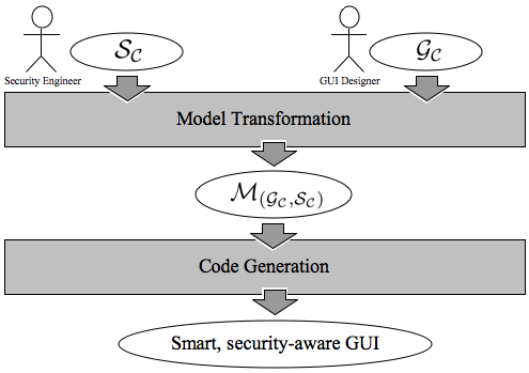

Figure 1: Modeling a smart and security-aware GUI.

to check that they are designing the right GUI to give the (authorized) users access to the (intended) application data.

3. SSG: A SMART, SECURITY-AWARE GUI BUILDER

SSG is a development environment, built using the Eclipse framework, for generating smart, security-aware GUIs following the model-based approach described in Section 2. In what follows, we describe the plugins included in SSG. All of the plugins are publicly available at [3].

3.1 Data model editor

The data model GMF-editor allows users to graphically model application data. This editor supports a simple language, named ComponentUML, for modeling application data. Essentially, this language provides a subset of UML class models: entities can be related by associations and may have attributes. Hence, the editor provides a concrete graphical syntax for modeling entities, with their attributes and association-ends.

3.2 Security-design model editor

The security-design model GMF-editor allows users to model an application's access control policy. This editor supports a language, named SecureUML+ComponentUML [2], for modeling access control policies on ComponentUML resources, i.e., on entities, their attributes and associations. The policies that can be specified in SecureUML+ComponentUML are of two kinds: those that depend on static information, namely the assignments of users and permissions to roles, and those that depend on dynamic information. The actions that can be controlled in SecureUML+ComponentUML are, e.g., those to 'create' and 'delete' entities, and to 'read' and 'update' their attributes. SecureUML+ComponentUML also provides composite actions, which group primitive actions into a hierarchy of higher-level ones. The composite actions are 'read', 'update', and 'full access' either on entities or entity's properties: for example, 'full access' on an attribute includes both 'read' and 'update' access on this attribute.

3.3 GUI model editor

The GUI model GMF-editor allows users to model an application's graphical user interface. This editor supports a language, named GUI [1, 3], for modeling the behavioral properties of GUIs, namely what are the actions associated to the different events that are supported by the GUI. In a nutshell, this language can be used to model GUIs that consist of widgets (buttons, entries, labels) that are displayed inside containers (windows, combo-boxes), which are themselves widgets. Each widget has a set of events (e.g., on-click and on-create), associated to it. These are the events supported by the widget. Each event is associated with a set of actions: these are the actions triggered by the event. Events' actions are of two types: widget actions (which are actions on GUI widgets, e.g., open, close, focus, and set) and data actions (which are actions on the application data). Both widget and data actions may take parameters. Also, each container has a (possibly empty) set of variables associated to it: these variables hold information that can be used by actions within this container.

3.4 GUI model generator

This QVT-generator automatically transforms a GUI model and a security-design model (both sharing the same data model) into a model of a new GUI. The resulting model has the same behavioral properties as the one modeled by the given GUI model, except that it is now both smart and security-aware with respect to the access control policy modeled by the given security-design model. The new GUIs are modeled using a language, named SecureUML+GUI, that allows modeling the behavioral properties of GUIs along with the information about which role can execute which events on these GUIs.

3.5 Code generator

The JET-code generator automatically generates, from a smart, security-aware GUI model, a full web application consisting of a collection of PHP-web pages whose design and behavior implements those modeled by the given smart, security-aware GUI model. In particular, windows are implemented as web pages. Thus, opening a window is implemented as loading the corresponding page and closing a window is implemented as loading the previously visited page. More interestingly, data actions (like 'create' or 'delete' entities and 'update' or 'read' their attributes) are implemented as SQL queries or statements on a data-base implementing the underlying data model; for convenience, the code generator can also create this database for the user. Finally, permissions to execute events on widgets (like clicking a button or creating an entry or a text box) are implemented by appropriate conditional statements on the PHP-code responsible for interpreting those events.

4. REFERENCES

[1] D. Basin, M. Clavel, M. Egea, and M. Schläpfer. Automatic generation of smart, security-aware GUI models. In F. Massacci, D. Wallach, and N. Zannone, editors, *Proceedings of the 2nd International Symposium on Engineering Secure Software and Systems (ESSOS 2010)*, volume 5695 of *LNCS*, pages 201–217. Springer-Verlag, 2010.

[2] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15(1):39–91, 2006.

[3] B. S. Group. The SmartGUI Project. `http://www.bm1software.com/`, 2009.