Securing **FrameWeb**: Supporting Role-based Access Control in a Framework-based Design Method for Web Engineering

Rodolfo Costa do Prado  
Ontology and Conceptual Modeling Research Group (NEMO), Departament of Computer Science, Federal University of Espírito Santo (UFES), Brazil  
rodolfocostapr@gmail.com

Vítor E. Silva Souza  
Ontology and Conceptual Modeling Research Group (NEMO), Departament of Computer Science, Federal University of Espírito Santo (UFES), Brazil  
vitor.souza@ufes.br

ABSTRACT

FrameWeb is a method for the development of Web-based Information Systems whose architectures are based on popular types of frameworks, such as Front Controller, Dependency Injection and Object/Relational Mapping frameworks. Also commonly used, Security Frameworks provide role-based access control through authentication and authorization features that can be reused if properly configured. In this paper, we extend FrameWeb to support Security Frameworks, allowing developers to model the aforementioned features in architectural design models using a graphical editor and generating code for the configuration of the framework and related artifacts. The proposal is validated using the code generator and comparing with artifacts from real projects.

CCS CONCEPTS

- **Software and its engineering** → Software design engineering
- **Information systems** → Web applications;

KEYWORDS

Web Engineering, Frameworks, FrameWeb, Authentication, Authorization, Role-based Access Control, Code Generation

ACM Reference format:

https://doi.org/10.1145/3243082.3243092

1 INTRODUCTION

A few decades ago, the pressing need for disciplined approaches and new methods and tools for the construction of Web-based systems spawned the new research & development field of Web Engineering [12]. Many researchers responded to this call, proposing systematic approaches for the design of Web applications.

One such method is FrameWeb, the Framework-based Design Method for Web Engineering [15]. FrameWeb is based on the premise that most Web-based Information Systems (WISs) are usually developed on top of a solid Web infrastructure which commonly includes frameworks such as a Front Controller [1] to mediate communication between Web pages (front-end) and services (back-end), a Dependency Injection mechanism [7] to resolve dependencies among back-end components and an Object/Relational Mapping [2] solution to communicate with the database.

FrameWeb incorporates concepts from the aforementioned categories of frameworks into a set of architectural design models, improving developer communication and project documentation. It follows a model-driven approach [10], in order to allow developers to extend support to other framework instances within the supported categories, providing a graphical editor [3] that guides developers in the use of the method’s modeling language, and a code generator [4] that relieves programmers from much of the coding effort, allowing them to concentrate more on business logic and less on infrastructure.

Currently, the method supports only Front Controller, Dependency Injection and Object/Relational Mapping frameworks. A feature that is very commonly implemented in WISs using frameworks is that of authentication & authorization, or role-based access control. Security Frameworks such as Spring Security (https://projects.spring.io/spring-security/), Apache Shiro (https://shiro.apache.org) and the standard JAAS (Java Authentication and Authorization Services) all for the Java platform, implement access control features in a generic way, allowing developers to fill in the gaps with respect to the particular WIS being built, increasing developer performance (less code to write) and system reliability (the frameworks have been extensively tested).

In this paper, we propose to add support for Security Frameworks to FrameWeb, allowing developers to specify authentication & authorization features in architectural design models using a generic language, generating code to their framework of choice, thanks to FrameWeb’s extensibility characteristics. Such support is added to the method’s meta-models in order to extend its language syntax, then also implemented into the graphical editor and the code generator. The proposal is validated through the automatic generation of security-related code for real projects and comparing the results.

This paper is organized as follows: Section 2 introduces research used as baseline in the work; Section 3 presents the proposed changes to FrameWeb in order to support security frameworks; Section 4 reports on the evaluation of this extension; Section 5 discusses related work; finally, Section 6 presents the conclusions.
2 BASELINE

This section summarizes the FrameWeb method and the basic concepts of Role-based Access Control and Security Frameworks, on top of which we built our proposals in this paper.

2.1 The FrameWeb Method

FrameWeb [15] is a design method for the development of Web-based Information Systems (WISs) that assumes the usage of frameworks in the implementation of the WIS. Such frameworks are also taken in consideration when designing higher level artifacts with the goal of better guiding the implementation efforts.

The approach proposes a basic architecture which divides the system into three main tiers (Presentation, Business Logic, and Data Access) for better integration with three types of frameworks: Front Controller (e.g., JavaServer Faces/JSF), dependency Injection (e.g., Contexts and Dependency Injection for Java/CDI) and Object/Relational Mapping frameworks (e.g., Java Persistence API/JPA). These three tiers are then divided in five packages, namely:

- **View (Presentation)**: contains Web pages, stylesheets, client-side scripts and other user interface artifacts;
- **Control (Presentation)**: contains controller classes that handle the requests made in the View package, using the infrastructure of the Front Controller framework, and call services offered by the Application package;
- **Application (Business Logic)**: contains the classes that are responsible for implementing the system’s functionalities, whose dependencies (with Control and Persistence) are wired by the Dependency Injection framework. Application classes manipulate objects from the Domain package and persist them via the Persistence package;
- **Domain (Business Logic)**: contains the classes that represent the domain of the application and annotations that guide the Object/Relational Mapping framework in persisting their data;
- **Persistence (Data Access)**: contains the DAO (Data Access Object) classes, responsible for the persistence, i.e., using the Object/Relational Mapping framework services for storing objects in the relational database.

Artifacts of the packages mentioned above are represented in four diagrams that can guide (and generate code for) the implementation of the system and the configuration of the different frameworks being used:

- **Persistence Model**: represents the classes from the Persistence package, showing which data access methods exist for each domain class;
- **Entity Model**: represents classes from the Domain package and their object/relational mapping meta-data;
- **Application Model**: represents classes from the Application package and their dependency network, i.e., which classes from Persistence they depend on and which classes from Control depend on them;
- **Navigation Model**: represents the components that form the presentation layer (View and Control packages) and how they interact.

Following a model-driven approach [13], FrameWeb provides a meta-model [10] that defines the abstract syntax for the above models, reusing the concrete syntax from UML. Figure 1 shows the Navigation Model for an example login application implemented with JSF, extracted from an on-line tutorial\(^1\) and built using the FrameWeb Editor [3], a CASE tool that is based on the FrameWeb meta-model and guides developers in the syntax of the language.

The model represents two web pages (\(<\text{page}\>\) stereotype), each of them with a form (\(<\text{form}\>\) stereotype) and both interact with a single controller class (no stereotype). The loginForm in the page index.xhtml contains a few JSF components, among which the login.user inputText and the login.pwd inputSecret. Once submitted, their values are bound to the user and pwd attributes of the login attribute in the LoginController object, followed by the execution of the validate() method in that object. Dependency relations between the controller and the Web pages show that two outcomes can follow: result=failure will take the user back to index.xhtml, whereas result=success will direct her to success.xhtml. A logout feature is also depicted.

The FrameWeb Editor [3], implemented using Eclipse Sirius,\(^2\) allows developers to import framework definitions in order to use specific constructs and rules from different frameworks. For instance, to build the model in Figure 1 the framework definition for JSF (that contains, among other things, the different visual components offered by the framework) is imported. Developers can further extend the editor in order for it to support other framework instances, while maintaining the same basic syntax for the models (given these framework instances belong to the categories of frameworks supported by the method).

Finally, the FrameWeb Code Generator [4] can generate skeletons of code artifacts (such as Web pages and classes) using the models built in the FrameWeb Editor as input. Extensibility is also featured here, as templates of code are also provided in the framework definitions, allowing the generator to be extended to support other

\(^1\)http://www.thejavageek.com/2013/12/18/login-application-jsf/.

\(^2\)https://www.eclipse.org/sirius/.
A Security Framework provides as reusable infrastructure a set of features concerned with the security of an application, such as authentication, authorization, cryptography, session management, etc. Here, we focus on authentication, i.e., certifying that a user is who she says she is; and authorization, i.e., verify if the user has the right to perform an action, given her authenticated credentials.

For authentication, we need to provide the framework instructions for the security in the application and which permissions are required for each action. This can be done in the Presentation layer by using the credentials informed are compared with the ones provided in a database. In the configuration of these frameworks, the field names for the credentials are defined and must be used in the login form. The form’s action property has to point to a specific URL, so when the login request is made, the Security Framework intercepts the request and performs the authentication.

Listing 1 shows a simple login form for Spring Security, which is configured with the contents of Listing 2. The configuration file specifies the URLs for login and login processing, as well as the names of the form fields to be used. This allows the framework to intercept the request sent by the login form, extract the values of the username and password fields and perform the authentication.

The credentials informed are compared with the ones provided by the application, which can also be defined in the framework’s configuration files, but for large-scale WISs it is more common to have some form of user database or LDAP integration in which a username and password can be persisted and accessed. That concept is named User Store and Security Frameworks interact with it in different ways, but the basis of verifying credentials is the same.

Beyond authentication, we need to provide the framework instructions for authorization. In some examples found in the documentation of the aforementioned Security Frameworks, authorization is made by simply associating user to roles and roles to classes and methods. However, that goes against the RBAC model, which prescribes the association of each user to a list of roles, and each role to a list of permissions.

Once the User Store is defined, we need to inform what actions need security in the application and which permissions are required for each action. This can be done in the Presentation layer by...
restricting the access of given URLs to certain permissions, but
good practice dictates one should secure the services in the Business
Logic layer. This is done by associating permissions to classes or
methods in the Application package directly, using a configuration
file or annotations, the latter being more common as the developer
can understand more easily the structure of system permissions by
looking at the source code of the class alone.

Listing 3 shows an annotation that secures a method: only users
who have the PERM_PERSON_DELETE permission can execute this
method. An unauthorized access generates an exception, which can
then be dealt with by the Presentation layer.

As with other kinds of frameworks, unless we somehow docu-
ment security configurations as the ones illustrated above in the
WIS’ architectural design, the burden to come up with this configu-
rations and manually code it lies with the programmer. In the next
section, we propose an extension of the FrameWeb method that
allows software architects to specify authentication and authoriza-
tion features in design models and have FrameWeb generate most
of the code related to the Security Framework usage.

3 PROPOSAL
In this section, we present an extension to the FrameWeb meta-
model, as well as the FrameWeb Editor and its code generator, so the
method also contemplates concepts of Role-Based Access Control
(RBAC) and Security Frameworks.

3.1 New Language Constructs
We extend the FrameWeb language in order to specify the configu-
ration of authentication and authorization features from Security
Frameworks, using RBAC concepts, in its models. As illustrated in
Section 2.3, such configuration involves defining:

1. The domain classes that represent users, roles and permis-
sions for authentication (e.g., Listing 1);
2. Aspects of the Web pages and forms that will trigger the
authentication (e.g., Listing 2);
3. Which permissions are required by each service method
or entire classes (e.g., Listing 3).

The above definitions involve, respectively, the Entity Model,
the Navigation Model and the Application Model. Given its model-
driven characteristics [10], to extend the FrameWeb language we
modified the respective meta-models of the three involved models.
Figure 3 shows fragments of these three meta-models, focused on
the changes performed in order to support authentication and au-
thorization. The original meta-classes are depicted in green, pink
and cyan for the Entity, Navigation and Application models respec-
tively. Light-gray meta-classes are a product of this work.

RBAC concepts are represented in the Entity Model. Basically,
applications adopting RBAC need to implement at least three enti-
ties: User, Role and Permission, with many-to-many associations
between Users and Roles and between Roles and Permissions. In
the FrameWeb meta-model (Figure 3) the meta-class DomainClass
was extended in order to add constructs for these concepts as entity
classes: AuthUser, AuthRole and AuthPermission. An AuthAttribute
meta-class derives from DomainAttribute so authentication and
authorization-specific attributes can be added to these new classes,
such as: AuthUserName, AuthPassword, AuthRoleName and Auth-
PermName (permission name). These last two are also extensions
from the IdAttribute meta-class. This is because, to perform the
authorization, Security Frameworks check a permission name from
the database against another written in the annotation or config-
uration file, so it is very important that the names for roles and
permissions are unique and refer to a single role or permission.
Securing FrameWeb: Supporting Role-based Access Control in a Framework-based Design Method for Web Engineering

Figure 4: Entity model with authentication constructs.

Figure 4 shows an Entity Model using the new meta-classes. We can see that the choice of concrete syntax was to add stereotypes «AuthUser», «AuthRole» and «AuthPermission» to the entity classes that represent, respectively, Users, Roles and Permissions. To ease visualization, the editor further displays these classes in a different color than “regular” entity classes. Note that these classes may have any name the designer wants to use, as their function in the authentication process is defined by the stereotypes.

While the structural part of authentication is represented in Entity Models, its behavior is defined in Navigation Models, such as the one in Figure 5. The model represents the login page («auth-Page» stereotype), the form with fields for user credentials («auth-Form»), as well as processing («AuthMethod»), success («AuthSuccessUrl») and failure («AuthFailureUrl») URLs. In the meta-model (Figure 3), authentication page and form are derived from Page and UIComponent meta-classes, respectively. Success and failure URLs are represented by subclasses of ResultDependency, because of their dependency from the result of the login process method. The processing URL is actually represented by AuthProcessingMethod (a FrontControllerMethod), i.e., the URL that activates this method will be the processing URL. This way, the model is still relevant with an implementation of security using a custom service class to provide credentials and permissions, as some developers may need to further customize the security of their Web applications.

Finally, as explained in Section 2.3, authentication is configured in service methods, hence, in FrameWeb Application Models. Figure 6 shows an Application Model with permissions expressed using UML constraints as concrete syntax. Service class PersonServicelmpl requires a permission named PERM_PERSON to be accessed. Service methods delete() and update() further require permissions named PERM_PERSON_DEL and PERM_PERSON_UP, respectively. This is accomplished by extending the meta-classes ServiceClass and ServiceMethod into AuthServiceClass and AuthServiceMethod respectively (Figure 3). These subclasses both have an attribute called PermissionName, which holds the name of the permission required for access.

3.2 Tool Support

As shown by the figures in this section, which were built with the FrameWeb Editor [3], the changes in the meta-model were also incorporated into the editor. Extending the FrameWeb Editor meant creating container nodes, nodes and edges to visually instantiate the new meta-classes added to the FrameWeb meta-model. In Figure 7 we can see a portion of the Sirius tool in which these elements are defined. With these additions, we have superclasses and subclasses being instantiated in the same model, which causes the tool to create two nodes when depicting subclasses, one representing an instance of the subclass, another an instance of the superclass. To avoid that, the Acceleo Query Language\(^3\) was used in these nodes instead of a regular expression for deciding what elements to show, as depicted in Figure 8 (see Semantic Candidates Expression property). This guarantees that the node is only shown when exactly the superclass is created.

The models with authentication and authorization constructs, made in the architectural design phase, can be used by developers in the configuration of Security Frameworks, as the concepts represented in them are adopted by the main Security Frameworks (at least in the Java platform) and can provide the basis for the usage of these tools in a Web system. Better yet, developers can have the artifacts that contain such configurations automatically generated from the models, as described next.

\(^3\)AQL, http://www.eclipse.org/acceleo/documentation/.
3.3 Code Generation

The FrameWeb Code Generator [4] translates FrameWeb models into code, producing artifacts that a developer can use in the construction of a WIS. The code generator navigates the .frameweb models and extracts the information required for creating project classes, pages and other files. With the extensions described in Section 3.1, the tool was also extended to understand the authentication and authorization concepts required for minimal usage of the Security Frameworks considered in this paper.

Listing 4 shows an excerpt of a .frameweb file depicting the Navigation Model presented at Figure 5 in XML format. Taking the processing URL as example, to get that information from the model the tool searches for the packagedElements tags with the type FrontControllerClass inside the packagedElement of type ControllerPackage, which results in a list of controllers. Inside each FrontControllerClass, the type AuthProcessingMethod is searched. When found, the name property of the AuthProcessingMethod is used to build the login processing URL. The same approach is used to get the login page URL, searching the AuthPage type. As for the success and failure URLs, the respective page is found in the client property that both types inherit from the ResultDependency meta-class. For instance, the code generator looks for a packagedElement with type AuthSuccessUrl and reads the client property in order to build the authentication success URL.

Listing 5 shows a template that serves as input for the tool to generate an XML configuration file for Spring Security. The Code Generator replaces specific tags (e.g., FW_AUTH_LOGIN_PROC_URL) with the information extracted from the model (e.g., the processing URL), as explained above. The template makes sure that the login page URL and login processing URL are accessible to unauthenticated users, so the authentication can be possible.

Listing 6 shows the Entity Model portion of the .frameweb file. The tool searches for the AuthUser, AuthRole and AuthPermission types. Inside AuthUser it looks for ownedAttribute tags with type AuthUserName and AuthPassword in order to match against the
Securing FrameWeb: Supporting Role-based Access Control in a Framework-based Design Method for Web Engineering

### Listing 6: Excerpt of a framework file for an Entity Model.

```xml
<compose xsi:type="frameweb:EntityModel" name="Entity Model">
  <packageElement xsi:type="frameweb:DomainPackage" name="Entity">
    <packageElement xsi:type="frameweb:AuthUser" name="User">
      <ownedAttribute xsi:type="frameweb:AuthUsername" name="username" />
      <ownedAttribute xsi:type="frameweb:AuthPassword" name="password" />
      <ownedAttribute xsi:type="frameweb:IdAttribute" name="user_id" />
    </packageElement>
    <packageElement xsi:type="frameweb:AuthRole" name="Role">
      <ownedAttribute xsi:type="frameweb:AuthRoleName" name="roleName" visibility="private" type="//@compose.3/Entity%20Package/String" />
    </packageElement>
    <packageElement xsi:type="frameweb:AuthPermission" name="Perm">
      <ownedAttribute xsi:type="frameweb:AuthPermName" name="permName" />
    </packageElement>
    <packageElement xsi:type="frameweb:DomainClass" name="Person">
      <ownedAttribute xsi:type="frameweb:IdAttribute" name="id" />
      <ownedAttribute xsi:type="frameweb:AuthUserName" />
      <ownedAttribute xsi:type="frameweb:AuthPassword" />
      <ownedAttribute xsi:type="frameweb:Email" />
    </packageElement>
  </packageElement>
</compose>
```

### Listing 7: Fragment of the generated xml configuration file for Spring Security with the SQL query for acquiring user credentials and permissions.

```xml
<authentication-manager>
  <jdbc-user-service data-source-ref="dataSource" users-by-username-query="select FW_AUTH_USER.
    , FW_AUTH_USER_USERNAME
    , FW_AUTH_USER_PASSWORD
    FROM FW_AUTH_USER
    where FW_AUTH_USER.FW_AUTH_USER_USERNAME = ?" authorities-by-username-query="select distinct FW_AUTH_USER.
    , FW_AUTH_USER.USERNAME
    , FW_AUTH_ROLE.FW_AUTH_PERM.
    , FW_AUTH_PERM.FW_AUTH_PERM_NAME
    FROM FW_AUTH_USER
    , FW_AUTH_USER_FW_AUTH_ROLE
    , FW_AUTH_ROLE
    , FW_AUTH_PERM
    where FW_AUTH_USER.FW_USER_ID = FW_AUTH_USER_FW_AUTH_ROLE.FW_USER_ID
    and FW_AUTH_ROLE.FW_AUTH_ROLE_ID = FW_AUTH_ROLE.FW_AUTH_ROLE_ID
    and FW_AUTH_PERM.FW_AUTH_PERM_NAME = ?" />
</authentication-provider>
</authentication-manager>
```

credentials provided in an authentication request. AuthRole and AuthPermission and their owned attributes AuthRoleName and AuthPermName will be used to get the list of permissions related to those credentials. Listing 7 shows a template for a Spring Security XML implementation that uses information extracted from the Entity Model to generate the code artifact.

Code generation based on the Application Model follows the same principles and is not detailed here due to space constraints. Further, different templates could be provided in order to generate code for different frameworks (e.g., Apache Shiro, JAAS, etc.).

### Table 1: Results of evaluation using Web Development projects.

<table>
<thead>
<tr>
<th>Project</th>
<th>Auth?</th>
<th>RBAC?</th>
<th>Gen</th>
<th>Not</th>
<th>Adj</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGAF</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>37</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>C2D</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>20</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>S2C-VV</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>37</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Implementation described in this paper is available at a public source code repository\(^1\) for the interested reader.

### 4 EVALUATION

The main purpose of this work is to simplify the configuration of security frameworks through graphical models, plus generate code for such configuration. With that in mind, to evaluate our work we used projects developed by undergraduate students from our university in the context of a Web Development course\(^2\) and measured the extent to which FrameWeb models could have helped developers implement Role-Based Access Control (RBAC). We report on the result for three projects in particular: SGAF (publication of movie reviews), C2D (evaluation of researchers based on their publications) and S2C-VV (management of religious activities).

The following methodology was used: (1) the security level present in the original application was identified; (2) models were designed using FrameWeb Editor; (3) the models were used to generate code for a Security Framework; (4) the code lines generated were compared against the code lines already present in the application’s security files, if any, in order to verify the coverage of what was generated; (5) the number of adjustments needed to be made in the generated code in order to prepare configuration files to implement RBAC was measured.

Table 1 shows the result of the evaluation, with the first columns reporting on the security level analysis for each application. Column Auth? indicates whether the original project implemented authentication / authorization, whereas column RBAC? reports if true RBAC was already implemented, according to the definition presented in Section 2.2.

The remaining columns show the comparison of the generated code with the original. Column Gen informs the number of lines automatically generated to be used in the Security Framework configuration. Column Not indicates how many lines related to security features were present in the original projects but were not automatically generated by FrameWeb, such as security database definitions, hash configuration, etc. Column Adj depicts how many lines were generated, but required manual adjustments, like indicating the basic permission name, required for a secured JAAS implementation. Finally, column % shows the percentage of lines of code generated over the total lines required to implement security in these projects.

These results tell us that FrameWeb was able to generate most of the code needed to implement security features based on the proposed extensions to its models, relieving the developers of most of the coding effort in this context. Moreover, the code generated

---

\(^1\)https://github.com/nemo-ufes/FrameWeb.

\(^2\)https://github.com/dwws-ufes/.
by FrameWeb implements true RBAC, which was not present in any of the projects reported in Table 1.

However, it is important to mention that the evaluation presented here is preliminary. It is currently under planning a new evaluation effort with practitioners, following a more formal protocol, in order to try and assess if the benefits of communicating through models and generating code outweigh the learning curve to use FrameWeb and its editor. Further efforts, regarded as future work, could evaluate the approach on nontrivial applications, in industrial settings, comparing different security frameworks, etc.

5 RELATED WORK
As mentioned in Section 1, the FrameWeb method aims to visually depict concepts of the frameworks that will be used in the construction of Web-based Information Systems (WISs). Previous work in the context of FrameWeb [4, 10, 15] have not proposed the addition of Security Frameworks to the list of supported types of framework. Our work aims to fill this gap. Other works in the literature have also proposed modeling security features for WISs, as follows.

SecureUML [9] proposes a UML-Based modeling language to create models that depict the authorization structure intended for an application. Like FrameWeb, the SecureUML meta-model is derived from the UML meta-model and adds concepts of RBAC (cf. Section 2.2). In SecureUML models, instances of Role are represented and a Permission is depicted as a relation between a specific role and a class, associating that relation with a certain action. Possible security constraints can be added in the model as UML constraints bound to classes or permissions relations.

The work presented in this paper differs from SecureUML because it does not aim to represent instances of roles or users. The only RBAC concept that has instances represented in our work is that of permissions, which is included in the Application Model (cf. Section 3.1). This addition to FrameWeb depicts the classes that will be instantiated into roles, users and permissions. This way, the users of an application can dynamically create their own authorization scheme. A SecureUML model would be a photograph of that scheme, although concepts presented in [9], such as role hierarchy and security constraints, could be added to FrameWeb in the future.

Pavlich-Mariscal et al. [14] propose an extension to the UML language to model the security aspects of an application domain, which can be really useful for creating the authorization requirements of a system. However, it is also a static representation of this authorization logic, whereas the work in this paper aims to model how a Security Framework can be used to enforce that logic.

UMLsec [8] also extends UML to create models with the intent of simplifying the development of security critical systems, but the security scope of the work is much larger than the one presented in this paper, taking in consideration the environment of an application whereas this work is focused on the design and implementation of WISs that use Security Frameworks.

Emig et al. [5] propose a metamodel for access control for Web service architectures that extends the RBAC meta-model, but it does not takes in consideration the usage of frameworks to handle security, so the models created would not be as useful to guide implementation in this particular architectural style or even used for automatic code generation.

6 CONCLUSIONS
In this paper, we propose an extension of FrameWeb’s modeling language, graphical editor and code generator in order to support the specification of authentication & authorization features in architectural design models in a generic way and the generation of code for a specific Security Framework, improving developer communication and relieving programmers from most of the coding effort in this regard.

Our initial evaluations show promising results. However, future work includes further evaluation (with practitioners), experiments involving frameworks from platforms different than Java and investigating other features from Security Frameworks that might benefit from further extensions of the FrameWeb language.

ACKNOWLEDGMENTS
NEMO (http://nemo.inf.ufes.br) is currently supported by Brazilian research funding agencies CNPq (process 407235/2017-5), CAPES (process 23038.028816/2016-41), and FAPES (process 69382549/2015).

REFERENCES