The FrameWeb Approach to Web Engineering: Past, Present and Future

Vítor E. Silva Souza[0000−0003−1869−5704]

Ontology & Conceptual Modeling Research Group (NEMO)
Department of Computer Science
Federal University of Espírito Santo (UFES) - Vitória, ES, Brazil
vitor.souza@ufes.br
http://www.inf.ufes.br/~vitorsouza/

Abstract. The use of software frameworks is a popular method of reuse, especially in the context of Web-based Information Systems (WISs) development, as they share a very similar basic infrastructure that is generalized and implemented in many different state-of-the-practice frameworks available. Moreover, there has been a growing interest in publishing data on the Web in a machine-processable format known as *linked data* in order to enable automatic processing of the huge amount of data available today on the Internet. The goal of the *FrameWeb* project is to provide methods, languages and tools to aid developers in the construction of WISs that take advantage of the architectural foundation offered by such frameworks and facilitate the publication of linked data from such WISs. In this paper, we review the history of the *FrameWeb* project, describe the approach in its current (end of 2019) form and devise plans for its near future.
This paper has been written to honor Ricardo de Almeida Falbo on the occasion of his formal retirement as a professor of the Department of Computer Science at the Federal University of Espírito Santo (UFES). The research project described herein would not have existed without him. Hence, this paper shows the fruits that came from a seed planted by Ricardo more than a decade ago.

Keywords: Reuse · frameworks · Web Engineering · method · language · tools · FrameWeb

1 Introduction

Software reuse has been practiced since programming began, using, e.g., libraries, domain engineering, design patterns, componentry, etc. [15]. A popular method of reuse is the use of software frameworks (e.g., Hibernate [6]) or platform architectures (e.g., Java™ Enterprise Edition [11]), which are middleware on/with which applications can be developed [15]. The use of such frameworks[1] helps to avoid the continual rediscovery and reinvention of basic architectural patterns and components, reducing cost and improving the quality of software by using proven architectures and designs [25].

[1] In this paper, the term *framework* is used both in its traditional sense—a reusable set of libraries or classes for a software system—and in the sense of *platform architectures* mentioned above.

This is particularly evident in the context of Web-based Information Systems (WISs) development, which is the focus of this paper. WISs are data-centric applications deployed on the Internet or an intranet, in which functionality and data have greater importance than content presentation. Such systems are usually developed on top of a solid Web infrastructure which commonly includes a Front Controller [5], a Dependency Injection mechanism [14], an Object/Relational Mapping [6] solution to communicate with the database, and so on.

Despite their popularity, until recently, and to the best of our knowledge, none of the Web Engineering [22] methods and modeling languages proposed in the literature considered the existence of such frameworks before the coding phase of the software process. Given how these frameworks affect the architecture of a WIS, this fact motivated us to propose *FrameWeb*, a Framework-based Design Method for Web Engineering [27, 28]. *FrameWeb* incorporates concepts from well established categories of frameworks (such as the ones above) into a set of architectural design models, improving developer communication and project documentation.

On a different, but related front, an increasing number of people and organizations are choosing to share their data on the Web, contributing to a *data deluge*. This phenomenon creates problems such as how to provide access to data so it can be most easily reused; how to enable discovery of relevant data; or how to integrate data from different and formerly unknown data sources [18]. A solution that has been gaining momentum in recent years is the publication of *linked data* [7], a set of technologies that lay the foundation for what researchers have been calling *The Semantic Web* [8] for the past two decades.

According to the Semantic Web vision, making data available on the Web in such a machine-processable format, would allow the creation of software agents that could help us through the data deluge, executing tasks that are repetitive, impractical or even impossible to accomplish nowadays. One of the main issues with this vision is that the current level of adoption by data publishers and application developers is not enough for us to harness all the advertised benefits of this new *Web of Data*.

In this context, *FrameWeb* provides a systematic method based on well-founded conceptual models, coupled with tools that automate certain parts of the process, facilitating the task of integrating a WIS into the Web of Data and, thus, promoting the adoption of linked data. Although a small contribution regarding the broader problem of realizing the Semantic Web vision, we can nevertheless harness the benefits of linked data, even if such vision has not been (or will never be) reached.

Since its initial proposal [27, 28], the *FrameWeb* approach has evolved in a number of ways [21, 10, 9, 4, 24], involving many undergraduate and graduate students in a research project.[2] The goal of the *FrameWeb* project is to provide methods, languages and tools to aid developers in the construction of WISs that take advantage of the architectural foundation offered by such frameworks and facilitate the publication of linked data from such WISs.

In this paper, we review the history of the *FrameWeb* project in Section 2 — what was its initial proposal and how it evolved —, describe the approach in its current (end of 2019) form in Section 3 — what can the approach help me accomplish now — and

[2] See https://nemo.inf.ufes.br/projects/frameweb/.

devise plans for its near future in Section 4 — what can we expect as future work in the project. Finally, Section 5 concludes the paper with some personal notes.

2 Past: the FrameWeb Story

FrameWeb's initial proposal [27, 28], developed between 2005 and 2007, focused on three specific frameworks based on my previous experiences in developing Web-based Information Systems (WISs) in practice: Struts,[3] Spring[4] and Hibernate.[5] These frameworks established, later, the initial set of *framework categories* that *FrameWeb* would support:

- **Front Controller frameworks** (e.g., Struts): frameworks of this kind implement a slightly modified version of the Model-View-Controller pattern [16], adapted to the Web and are, thus, also known as *MVC frameworks*. When using such a framework, a WIS manages all requests from clients using an object known as Front Controller. Based on its configuration,[6] this object decides which class (called a *controller class*) will respond to the current request. Then, it instantiates an object of that class, calls one of its methods and, based on the method's return value, the Front Controller decides the appropriate view to present as result, such as a Web page, a PDF report, a file download, etc. For instance, in the Java EE set of standards, JavaServer Faces[7] is a Front Controller framework;
- **Dependency Injection frameworks** (e.g., Spring): frameworks of this kind allow the developer to program to interfaces, i.e., when classes depend on objects of other classes to perform a certain task, have the dependent class relate only to the interface of its dependencies, and not to a specific implementation of that service [14]. Such dependencies are specified in the framework's configuration and, when a certain object is created (which is also performed by the framework), all of its dependencies are automatically injected and satisfied. For instance, in Java EE, Contexts and Dependency Injection for Java[8] is the standard Dependency Injection framework;
- **Object/Relational Mapping frameworks** (e.g., Hibernate): frameworks of this kind offer automatic and transparent persistence of objects to tables of a relational database management system (RDBMS) using meta-data that describe the mapping between both worlds [6]. Such frameworks became very popular (and not

[3] https://struts.apache.org

[4] https://spring.io/projects/spring-framework

[5] https://hibernate.org/orm/

[6] Frameworks are usually configured using specific files or annotations in the classes themselves. Often, sensible defaults help keep such configuration to a minimum.

[7] JSF, http://jcp.org/en/jsr/detail?id=344. Strictly speaking, however, JSF (and the other Java EE standards) are *specifications* that can be implemented by many frameworks (e.g., Mojarra and MyFaces implement JSF). When using a Java EE certified application server, however, this is not explicit to the developer. As such, we will refer to these standard specifications as being frameworks themselves.

[8] CDI, http://jcp.org/en/jsr/detail?id=346

only on WISs) due to what has been called the *object-relational impedance mismatch* [19], i.e., a set of problems that arise due to the combination of the object-oriented paradigm (popular choice for software development) and the relational paradigm (popular choice for data storage). For instance, the Java EE standard for Object/Relational Mapping is the Java Persistence API.[9]

FrameWeb, thus, incorporates the concepts from these frameworks into design models. Initially, it started with two main contributions to the architectural design phase of the software process: (i) the definition of a basic architecture (detailed in Section 3) for better integration with these kinds of frameworks; and (ii) a UML profile (lightweight extension) for the construction of four different design models that bring the concepts used by the frameworks to the models.

Figure 1 illustrates some of the proposed extensions in a **Navigation Model**, which is the *FrameWeb* model that incorporates concepts from Front Controller frameworks. UML stereotypes are used to differentiate Web pages (⟨⟨page⟩⟩), templates (⟨⟨template⟩⟩, used to render Web pages), forms (⟨⟨form⟩⟩) and controller classes (no stereotype). Dependency associations with constraints indicate how the different components interact and, thus, guide the configuration of the Front Controller framework.

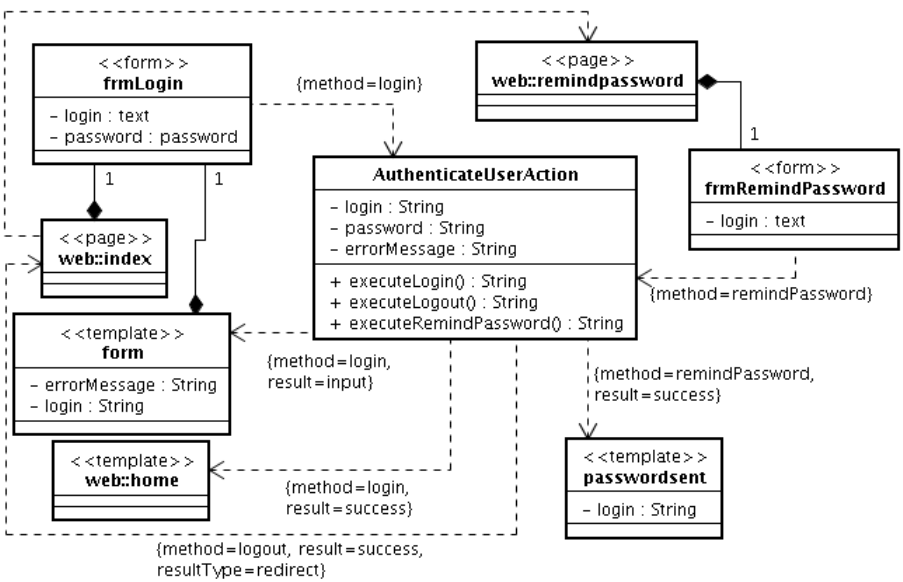

Fig. 1. Navigation Model for log in, log out and remind password features of a WIS [28].

This Navigation Model indicates that the index page of the WIS should have a form frmLogin with login and password fields, whose respective types (text and password)

[9] JPA, http://jcp.org/en/jsr/detail?id=338

refer to visual components from the *tag library* used by the framework. Once the user fills in and submits this form, the framework should respond with the AuthenticateUserAction controller, in particular its executeLogin() method (Struts suggested a standard execute prefix to all controller methods). If this method returned input (presumably due to some issue with the user input), the form template should render an error message related to the login attempt and the user may try again. Instead, if it returns success, the user should be directed to the home template. *Log out* and *remind password* features work analogously. Note that when components have attributes with the same name (e.g., frmLogin.login and AuthetnicateUserAction.login) it means that the framework should take care of this binding (e.g., have the contents of the login field in the form copied to the login attribute of the controller).

FrameWeb also prescribed three other models, all of them based on the UML Class Diagram: the **Domain Model** (later renamed **Entity Model**), the **Persistence Model** and the **Application Model**. It also offered an extension of the method, called *S-FrameWeb*, that prescribed the use of the Object Management Group's (OMG) Ontology Definition Metamodel (ODM) [1] in order to guide the creation of a vocabulary in OWL (W3C's Web Ontology Language)[10] representing the classes from the domain model of the WIS. Further, a component compatible with the Struts framework was implemented in order to output instances of this vocabulary based on the data from the WIS.

The original proposal of *FrameWeb* provided software engineers with interesting tools to organize and document the architecture of a WIS, giving precise instructions to developers on how they should write the code and configure the frameworks. However, it suffered from a few drawbacks:

(i) Its proposed models were based on specific instances of the supported framework categories (namely, Struts, Spring and Hibernate), with no guarantees they would fit appropriately if another set of frameworks (although from the same categories, say JSF, CDI and JPA) were used;
(ii) Although using UML lightweight extensions provides the advantage of allowing designers to use their UML case tool of choice, it does not prevent them from including UML constructs in the models that were not intended by the *FrameWeb* approach, or to use the ones that were intended, but in an inappropriate way;
(iii) Further, general-purpose UML tools will not validate the specific rules proposed by the *FrameWeb* approach for its models, nor provide code generation support for the kind of application that these models represent (e.g., web::index in Figure 1 would be generated as a class, not a Web page).
(iv) Finally, the method is focused on a particular architecture for WISs and the state-of-practice on Web development has evolved, producing different architectures (which use different kinds of frameworks), e.g., Progressive WebApps, Single-Page Applications, the use of microservices and front-end frameworks, etc.

Using Model-Driven Development (MDD) [23] techniques, we thus formalized a domain-specific language for *FrameWeb* models, whose abstract syntax is the metamodel illustrated (at a high-level of abstraction) in Figure 2. We decided to keep the concrete syntax the same as before, as the UML is a language that is familiar to most

[10] https://www.w3.org/TR/2012/REC-owl2-overview-20121211/

software developers. Hence, our meta-model depends on a *Partial UML Meta-model*, which contains the parts of UML that are used by *FrameWeb*. The meta-model is then divided in five components, one for each of the proposed *FrameWeb* models and a *Framework Meta-model* component, which allows us to specify rules and modeling constructs that are specific to the set of frameworks with which a given WIS will be built [21].[11]

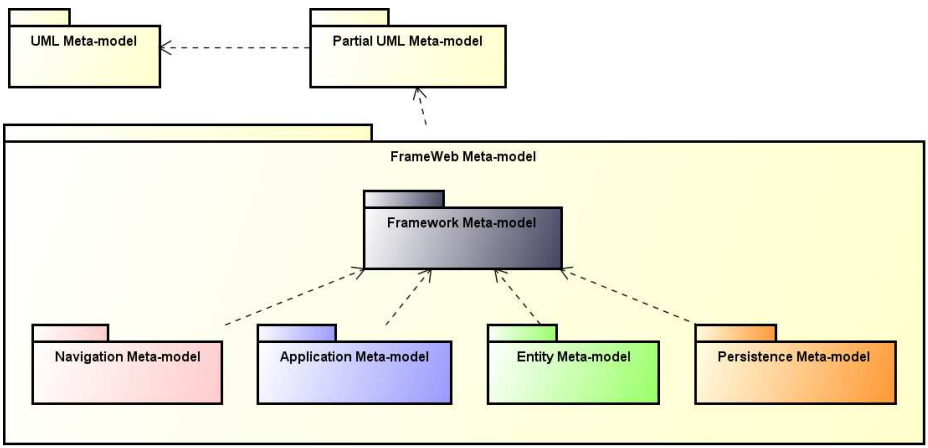

Fig. 2. Overview of the meta-model that defines the *FrameWeb* language [20].

Next, we proposed to replace the *S-FrameWeb* extension with a new one we called *FrameWeb-LD*. The new extension suggested the use of the higher-level, well-founded language OntoUML [17] for ontology capture and formalization and replaced the primitive linked data support that had been specifically built for the Struts framework with the Ontology-Based Data Access (ODBA) tool D2RQ,[12] which serves as a linked data adapter layer over the relational database already in use by the WIS. *FrameWeb-LD* extended the *FrameWeb* meta-model, providing new constructs that allow designers to link concepts of the WIS's domain with external linked data vocabularies, an important step to connect one's data to the Web of Data (or Semantic Web). Further, it provided a tool to help developers generate the operational ontology schema and the ODBA configuration [10].[13]

Once the *FrameWeb* language was defined, it became possible to build tools based on it. Using the infrastructure provided by the Eclipse Modeling Framework (EMF) [29] and the Sirius project [30], a first version of the *FrameWeb Editor* was developed. Figure 3 shows the top-level view of a *FrameWeb* model open in the editor. At the right-hand side a palette provides designers only with the constructs that are allowed in the model being edited; at the bottom, a list of properties allows one to set the attributes

[11] This is the result of the work of Masters student Beatriz Franco Martins Souza.
[12] http://d2rq.org/.
[13] This is the result of the work of Masters student Danillo Ricardo Celino.

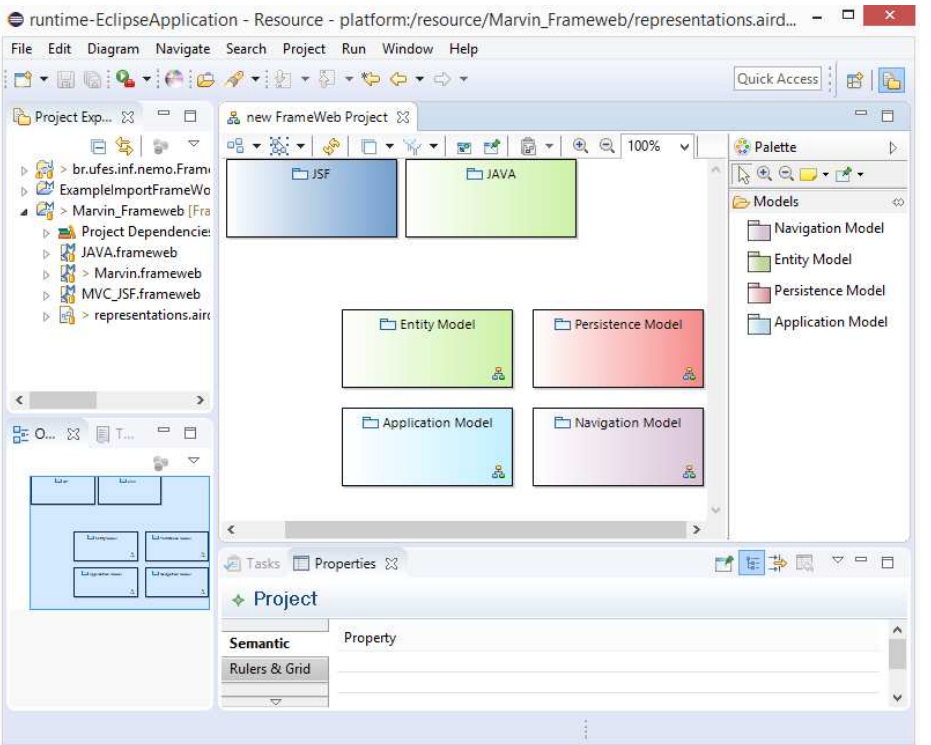

Fig. 3. The first version of the *FrameWeb Editor* [9].

of different model components; at the left-hand side, an overview of the project; in the center, the model being edited, in this case the project overview. Double-clicking one of the model components opens it for edition. Note how two platform-specific settings (Java and JSF) have been imported in the project, showing how the editor supports the extensibility of the method [9].[14]

Another tool built on top of the *FrameWeb* meta-model is the *FrameWeb Code Generator*. This tool reads the model created with the *FrameWeb Editor* and, for each element that represents an artifact of code (e.g., controller classes and Web pages in Navigation Models such as the one in Figure 1), it uses a template for that particular artifact in the specific framework/platform of choice (e.g., a controller class in JSF), filling in the blanks with data extracted from the model (e.g., the controller's name, attributes, methods, etc.) [4, 32]. After the first version of the *FrameWeb Code Generator* was implemented and integrated with the *FrameWeb Editor*, both tools were converted into Eclipse plug-ins with the purpose of integrating them with Web development projects in this IDE. A developer can now turn on the *FrameWeb* facet in their regular Eclipse

[14] This is the result of the work of undergraduate student Silas Louzada Campos.

project, design the models and have the code generated directly into the project structure.[15]

FrameWeb has also evolved in the direction of supporting new categories of frameworks. A feature that is very commonly implemented in WISs using frameworks is that of *authentication & authorization*, or *role-based access control*. To add support for Security Frameworks to *FrameWeb*, its meta-model was modified in order to extend its language syntax, with modifications also implemented in the graphical editor and the code generator. This now allows 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 [24].[16]

The evolution of *FrameWeb* so far has scratched the surface regarding the aforementioned drawbacks of the method. With respect to not being generic enough (drawback (i)), the definition of the *FrameWeb* language using MDD techniques has improved the method in this sense, but further studies (discussed in Section 4) are necessary to assure that the proposed language is, in fact, generic considering the supported categories of frameworks. On the other hand, such language definition solves drawback (ii), as developers now have a clear specification of how to write a *FrameWeb* model. Related to that, tool support (drawback (iii)) has also improved, but is an ongoing work that needs further development and polishing (part of which probably should take place outside Academia to guarantee a minimum level of quality required by the Industry). Finally, *FrameWeb* is still very much focused on a particular architecture (drawback (iv)). The inclusion of a new category of framework paves the way for further modification of the method's modeling language in order to support further kinds of frameworks and, as a later step, different WIS architectures.

3 Present: Developing WISs with FrameWeb

In this section, we describe what can be accomplished with *FrameWeb* at the moment (December 2019). Both the method and its tools are constantly being developed, thus some of the contributions described in the previous section have yet to be incorporated into the IDE (Integrated Development Environment).

In what follows, we first present what can already be done with the aid of *FrameWeb* tools (Subsection 3.1), then we talk about two features that have not yet been integrated: support for security frameworks (Subsection 3.2) and linked data publication (Subsection 3.3).

3.1 Tool-supported WIS Development

With *FrameWeb*, one can design Web-based Information Systems (WISs) that fit into the architecture shown in Figure 4. Based on the Service Layer pattern [13], this architecture divides the system in three layers: presentation, business and data access.

[15] This is the result of the work of Masters student Nilber Vittorazzi de Almeida and undergraduate students Breno Leite Zupeli and Lucas Ribeiro Mendes Silva.

[16] This is the result of the work of Masters student Rodolfo Costa do Prado.

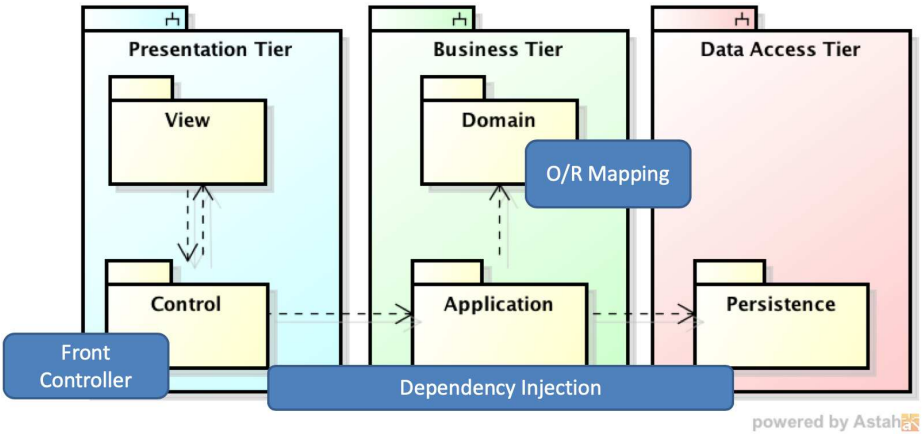

Fig. 4. *FrameWeb*'s supported architecture.

In the Presentation Tier, the View package holds the Web pages, stylesheets, client-side scripts and other user interface artifacts. At the Control package, controller classes handle the requests made by components of the View package, using the infrastructure of the Front Controller framework, and call services offered by the Application package.

In the Business Tier, the Application package 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. The Domain package contains the classes that represent the problem domain, plus annotations that guide the Object/Relational Mapping framework in persisting their data.

Finally, the Data Access Tier consists solely of the Persistence package, which contains the DAO (Data Access Object [5]) classes, responsible for the persistence, i.e., using the Object/Relational Mapping framework services for storing/retrieving objects in/from the relational database. This last tier/package is optional and its responsibilities could be merged into the Application package if desired. However, concentrating all data access operations regarding a given domain class into a single DAO class (which is the essence of the DAO pattern) helps with the maintainability of the code.

To develop a WIS with *FrameWeb*,[17] we should begin by installing its tools, as follows: first, install Java and the Eclipse IDE for Java EE Developers; then, install Sirius through the *Eclipse Marketplace* that can be accessed inside the IDE; finally, using the *Install New Software* feature of Eclipse and pointing it to the *FrameWeb* plug-in update site, install the Code Generator and the Graphical Editor *FrameWeb* tools.

Once the tools are installed, we can create a regular Web project in Eclipse, using the frameworks of our preference. In this section, we will illustrate the use of *FrameWeb*

[17] More detailed instructions can be found in a tutorial that can be accessed through the project's website: https://nemo.inf.ufes.br/projects/frameweb/.

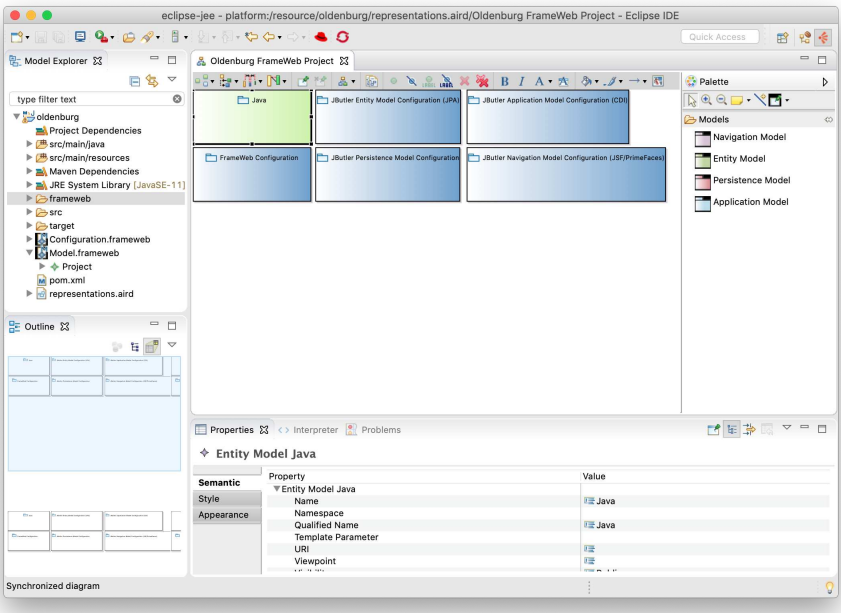

Fig. 5. A project in the Eclipse IDE with the *FrameWeb Editor* facet activated.

with a simplified conference management system,[18] focusing on a single functionality, namely: author registration. Once the project is created, we need to activate the *FrameWeb Editor* facet for that project, which will result in the inclusion of a blank *FrameWeb* model and configuration in that project. Switching to the Sirius perspective, those models can be opened in the *FrameWeb Editor*, as demonstrated in Figure 5.

At the top-right corner of the figure we can see that the Sirius perspective is active. At the left-hand side, the *Model Explorer* view shows our project's files. The files *Configuration.frameweb* and *Model.frameweb* were created when the *FrameWeb* facet was activated. By expanding the latter and double-clicking the *Project* item in the *Model Explorer*, we open the *FrameWeb Editor* in the center of the IDE, as shown. At the right-hand side, the palette allows us to create the four kinds of model the method supports.

The boxes that are already in the model represent *Architecture Definition Files* (divided in *Language* and *Framework Definition Files*) that were imported to the project before the screenshot of Figure 5 was taken. As previously shown in Figure 2, the *FrameWeb* language defines a *Framework Meta-model* component, which allows us to use,

[18] We envisioned a WIS that could be used by professors of Research Methodology classes to simulate a conference-like setting in which students could peer-review each others' papers, like a simplified EasyChair (https://easychair.org). We called it *Oldenburg*, in honor of the philosopher who is seen as the 'father' of modern scientific peer review, according to Wikipedia.

in our models, elements that are specific to the chosen platform/frameworks. At the project's source code repository,[19] we can copy a *Language Definition File* from the languages folder and a set of *Framework Definition Files* from the frameworks folder into our Eclipse project and the *FrameWeb* tools will automatically include them in the model.

In practice, *Language Definition Files* include a list of primitive types and classes from the API of the programming language of choice (e.g., Java has int, double, String, etc.) to be used as types of attributes and parameters in different models. In turn, *Framework Definition Files* include tags from the visual component library of choice (e.g., JSF component library PrimeFaces[20] has dataTable, inputText, password, etc.) to be used in Navigation Models and templates for code generation. For every combination of frameworks we want to use, a set of files should be created and imported. For instance, the frameworks/jbutler folder at the source code repository offers definition files for projects that use the JButler[21] utility classes together with the Java EE standards JSF, CDI, JPA and visual component library PrimeFaces.

We now demonstrate the design of the author registration feature of our running example. Figure 6 shows the Entity Model with the Author class and its object/relational mappings. Most of the mapping relies on sensible default values (e.g., table names are the same as class names, column names are the same as attribute names, column types are inferred, etc.), but string size and date precision are explicitly specified. The diagram does not show any ID or versioning (optimistic locking) attribute because they are inherited from a JButler utility class.

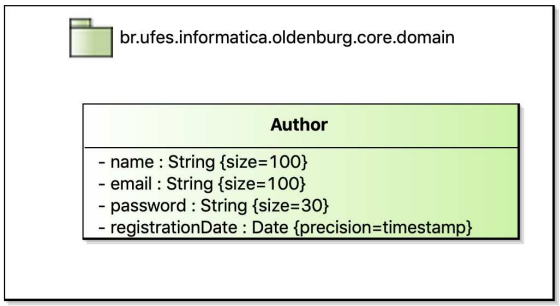

Fig. 6. *FrameWeb* Entity Model for our running example.

The persistence of objects of the Author class is handled by the AuthorDAO, shown in Figure 7. Most of the basic persistence operations (e.g., retrieve all, retrieve by id, save, delete, etc.) are inherited from a JButler utility class, therefore are not shown. The DAO is divided into interface and implementation, and the semantics of the *FrameWeb* language states that the former should declare the signatures of all public methods of the

[19] https://github.com/nemo-ufes/FrameWeb
[20] http://primefaces.org
[21] https://github.com/dwws-ufes/jbutler

latter, allowing us to use a simplified notation for the interface. By relying on JButler for the basic operations, the DAO only shows a method that is specific to our WIS: retrieving an author given her e-mail, required to check if someone is registering with an e-mail that has already been used.

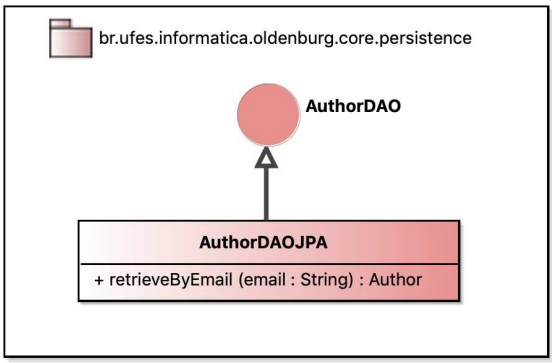

Fig. 7. *FrameWeb* Persistence Model for our running example.

The author registration feature is represented in the Navigation Model of Figure 8. Web pages in the core/registration/ path are used in this scenario, starting with the index page, which contains the registrationForm with inputText and password fields (from PrimeFaces). Once the form is submitted, the Front Controller copies the contents of the fields to attributes of RegistrationController (note that the fields with author. prefix are copied to internal attributes of the author object in the controller) and the register() method is called. Depending on the outcome, the user may be presented the success or the emailinuse pages, which require that the Front Controller bring some data (author.name and author.email respectively) back to the view.

Finally, the Application Model shown in Figure 9 completes the architecture with the RegistrationService which, like the DAO before, is divided in interface and implementation. The RegistrationController from the Navigation model depends on this service which, in turn, depends on the AuthorDAO to properly perform its register() method. The Dependency Injection framework will satisfy both dependencies when needed.

Once the models have been created, we can generate code for it. When doing it for the first time, we should click on the FrameWeb Configuration item of our project in the editor (as shown in Figure 5) and set a few properties, such the as *Class* and *Page Extensions* (e.g., .java and .xhtml), the *Framework Definition Path* — where the code generation templates are located — and the *Src* and *View Paths*, which is where classes and Web pages, respectively, will be generated. After that, right-clicking any blank space in the *FrameWeb Editor* and selecting *Generate Source Code* will create all the classes and Web pages from our models right into the structure of our project in Eclipse itself.

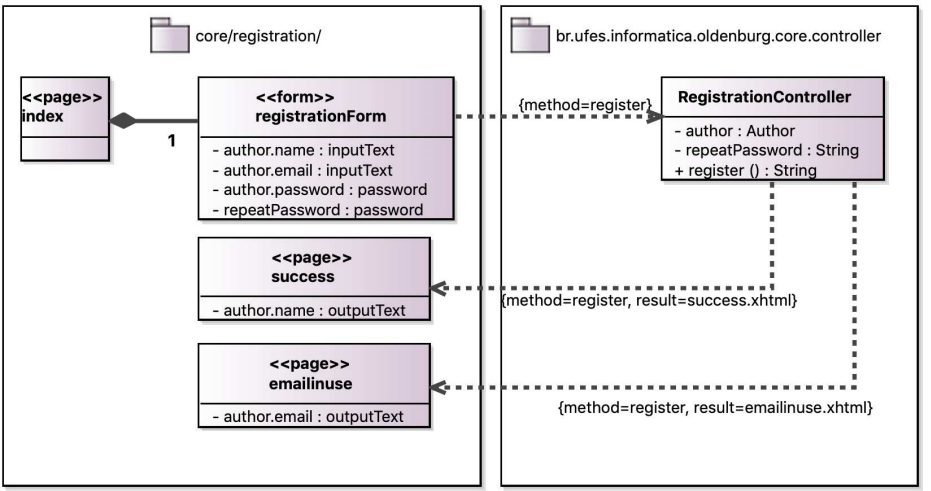

Fig. 8. *FrameWeb* Navigation Model for our running example.

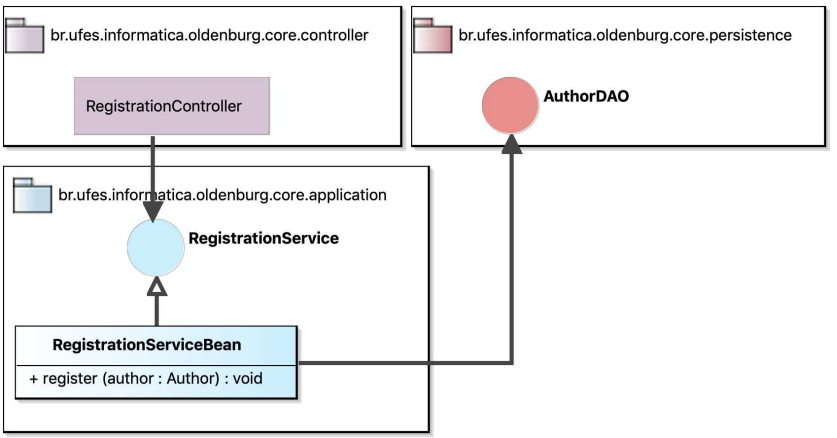

Fig. 9. *FrameWeb* Application Model for our running example.

Listing 1 shows one of the templates used in our running example. Code between {{ and }} are replaced by elements extracted from the Navigation Model. Further, {% and %} can be used to insert control flow directives like loops and conditionals in the template. The rest of it is standard Java code that defines a class that extends JSFController from JButler, is annotated with @Model to be referred to in Web pages, has association with service classes annotated with @EJB (dependency injection annotation), attributes defined with their respective accessor/mutator (get/set) methods and the skeletons (stubs) for other methods the controller might have. The result of applying such template to the Navigation Model of Figure 8 (plus running *Organize Imports* and *Format* features from Eclipse) is shown in Listing 2.

We can see that, apart from comments, we only need to implement the register() method to complete this particular artifact of code. Although further experiments are in order, recent tests with a different set of frameworks showed that the *FrameWeb Code Generator* generated between 68% and 94% of the lines of code of a simple WIS when compared with the final solution, after manual editing [26]. We consider this a good result in terms of cost (of modeling) vs. benefit (of less code to write).

3.2 Role-based Access Control

One recent *FrameWeb* extension that has not yet made its way to the Eclipse plug-in is the support for security frameworks that implement Role-based Access Control (RBAC). This extension of the *FrameWeb* language (i.e., meta-model) allows developers to specify authentication & authorization features in Entity, Application and Navigation models using a generic language and generating code to a security framework of choice.

Role-Based Access Control (RBAC) [12] is a basic model for authorization inside an application that is founded on the separation between *actors* and the *actions* available to them in the system. This separation is made by adding the concept of *roles*. In RBAC, any *permission* to run an action inside the application can only be associated with a role. Actors do not acquire permissions directly, instead they are given roles that aggregate a collection of permissions. With this configuration, the assignment of permitted actions to users inside a system is made with both simplicity and flexibility [24].

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. The proposed *FrameWeb* extension focuses 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. *FrameWeb* models can now define: (a) the domain classes that represent users, roles and permissions; (b) aspects of the Web pages and forms that will trigger the authentication; (c) which permissions are required by each service method or entire classes, to implement authorization.

Figure 10 shows an Entity Model that defines users (⟨⟨AuthUser⟩⟩ stereotype), roles (⟨⟨AuthRole⟩⟩) and permissions (⟨⟨AuthPermission⟩⟩) in a generic way, i.e., they could be used in or adapted to any WISs. For instance, to use them in *Oldenburg*, we could connect User to Author or have the latter annotated with ⟨⟨AuthUser⟩⟩ instead, use the author's email as the ⟨⟨AuthUserName⟩⟩, and so on.

Listing 1. Template for a controller class.

```
package {{ package.Name }};

import javax.ejb.EJB;
import javax.enterprise.inject.*;
import br.ufes.inf.nemo.jbutler.ejb.controller.JSFController;

/** TODO: generated by FrameWeb. Should be documented. */
@Model
public class {{ class.Name }} extends JSFController {
  /** Serialization id (using default value, change if necessary). */
  private static final long serialVersionUID = 1L;

  {% for association in associations %}
  /** TODO: generated by FrameWeb. Should be documented. */
  @EJB
  private {{ association.TargetMember.Type.Name }} {{ association.
       TargetMember.Type.Name | lower_first }};
  {% endfor %}

  {% for attribute in attributes %}
  /** TODO: generated by FrameWeb. Should be documented. */
  private {{ attribute.Type.Name }} {{ attribute.Name }};
  {% endfor %}

  {% for method in methods %}
  /** TODO: generated by FrameWeb. Should be documented. */
  {{ method.Visibility.Name }} {% if method.MethodType is null %}void{% else
       %}{{ method.MethodType.Name }}{% endif %} {{ method.Name }}({% for
       parameter in method.OwnedParameters %}{{ parameter.Type.Name }} {{
       parameter.Name }}{% if loop.last == false %}, {% endif %}{% endfor %})
       {
    // FIXME: auto-generated method stub
    return{% if method.MethodType is not null %} null{% endif %};
  }
  {% endfor %}

  {% for attribute in attributes %}
  /** Getter for {{ attribute.Name }}. */
  public {{ attribute.Type.Name }} get{{ attribute.Name | capitalize }}() {
    return {{ attribute.Name }};
  }

  /** Setter for {{ attribute.Name }}. */
  public void set{{ attribute.Name | capitalize }}({{ attribute.Type.Name }}
       {{ attribute.Name }}) {
    this.{{ attribute.Name }} = {{ attribute.Name }};
  }
  {% endfor %}
}
```

Listing 2. Generated code for a controller class.

```
package br.ufes.informatica.oldenburg.core.controller;

import javax.ejb.EJB;
import javax.enterprise.inject.Model;

import br.ufes.inf.nemo.jbutler.ejb.controller.JSFController;
import br.ufes.informatica.oldenburg.core.application.RegistrationService;
import br.ufes.informatica.oldenburg.core.domain.Author;

/** TODO: generated by FrameWeb. Should be documented. */
@Model
public class RegistrationController extends JSFController {
  /** Serialization id (using default value, change if necessary). */
  private static final long serialVersionUID = 1L;

  /** TODO: generated by FrameWeb. Should be documented. */
  @EJB
  private RegistrationService registrationService;

  /** TODO: generated by FrameWeb. Should be documented. */
  private Author author;

  /** TODO: generated by FrameWeb. Should be documented. */
  private String repeatPassword;

  /** TODO: generated by FrameWeb. Should be documented. */
  public String register() {
    // FIXME: auto-generated method stub
    return null;
  }

  /** Getter for author. */
  public Author getAuthor() {
    return author;
  }

  /** Setter for author. */
  public void setAuthor(Author author) {
    this.author = author;
  }

  /** Getter for repeatPassword. */
  public String getRepeatPassword() {
    return repeatPassword;
  }

  /** Setter for repeatPassword. */
  public void setRepeatPassword(String repeatPassword) {
    this.repeatPassword = repeatPassword;
  }

}
```

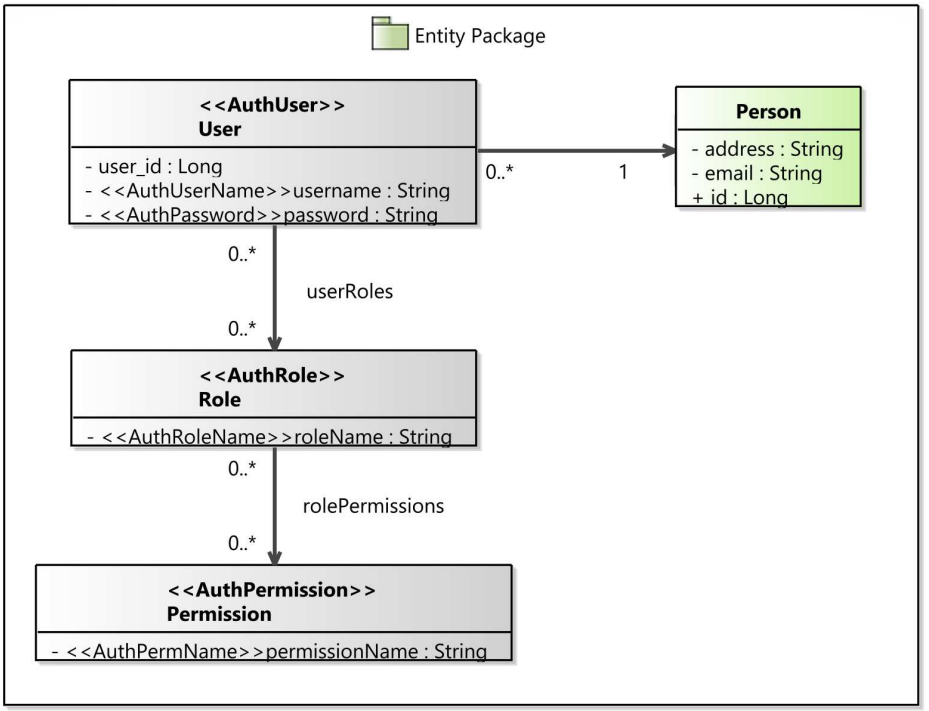

Fig. 10. *FrameWeb* Entity Model with RBAC features [24].

Figure 11 shows a Navigation Model that specifies how authentication will be implemented. The model represents the login page (⟨⟨authPage⟩⟩ stereotype), the form with fields for user credentials (⟨⟨authForm⟩⟩), as well as processing (⟨⟨AuthMethod⟩⟩), success (⟨⟨AuthSuccessUrl⟩⟩) and failure (⟨⟨AuthFailureUrl⟩⟩) URLs. This information will guide the security framework in performing authentication. Note that the processing URL actually refers to a method of the controller class so the security framework will use the URL that activates this method as the processing URL.

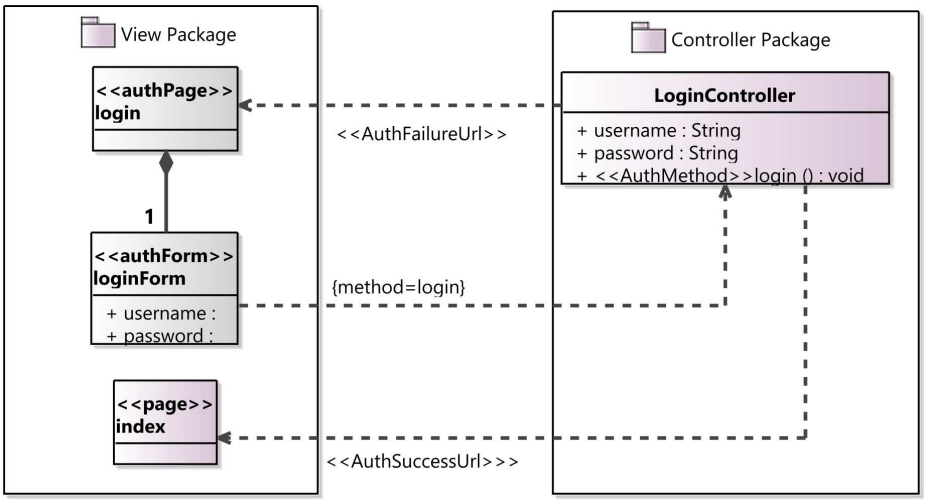

Fig. 11. *FrameWeb* Navigation Model with RBAC features [24].

Finally, Figure 12 shows an Application Model with authorization settings. Permissions are expressed using UML constraints as concrete syntax. Service class PersonServiceImp 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.

The RBAC extension for the *FrameWeb* method has been implemented, not only regarding the modifications in the meta-model but also with respect to tool support. Therefore, we can produce the models with security features using the *FrameWeb Editor*, as demonstrated by figures 10–12, and generate code with the *FrameWeb Code Generator*, as shown in [24]. However, this has been implemented on a separate code base,[22] and, thus, needs to be carefully merged into the code of the *FrameWeb* Eclipse plug-ins.

[22] https://github.com/Rodolfocostapr/Experimento-Frameweb-Sec

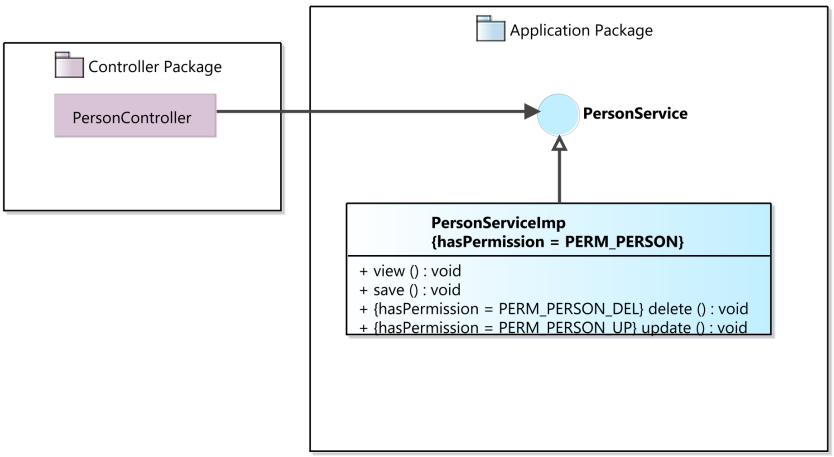

Fig. 12. *FrameWeb* Application Model with RBAC features [24].

3.3 Linked Data Support

Another extension of the *FrameWeb* language/meta-model that has not yet been incorporated into the *FrameWeb* tools is *FrameWeb-LD* [10]. Such extension allows developers to specify how the data from the WIS relates to well-known vocabularies from the Semantic Web, with the purpose of integrating them into the Web of Data [18]. Figure 13 shows an example of Entity Model with linked data mappings added to the domain classes.

The figure illustrates a system that manages researchers from a postgraduate program and their publications in order to produce reports on their research productivity. Although not shown in the diagram, vocabulary identifiers (IDs) are associated to their respective URIs, e.g., foaf is associated with http://xmlns.com/foaf/0.1/ (Friend of a Friend vocabulary) and dblp with http://dblp.rkbexplorer.com/id/ (DBLP Computer Science Bibliography dataset).

Then, concepts from external vocabularies are shown using their vocabulary IDs as UML namespace (e.g., foaf::Person). They can be related to classes from the WIS via UML associations, navigable towards the external class, representing an RDF triple: the class from the WIS is the subject, the external one is the object and the predicate is specified as a constraint. In the example, Researcher is owl:equivalentClass to dblp:Person. As a syntactic sugar, the rdfs:subClassOf relation between a class from the WIS and one from an external vocabulary can be represented by a UML inheritance association. In the example, Researcher is rdfs:subClassOf foaf:Person.

Triples concerning attributes of classes are represented using constraints in the form *predicate=object*. In the example, Researcher.name is owl:equivalentProperty to dblp:primaryFullPersonName. Constraints in associations between classes from our WIS establish relations among object properties (in the same way constraints in attributes establish relations among data type properties). In the example, the association between Publication and Venue is rdfs:subPropertyOf dblp:publicationType. Last, but

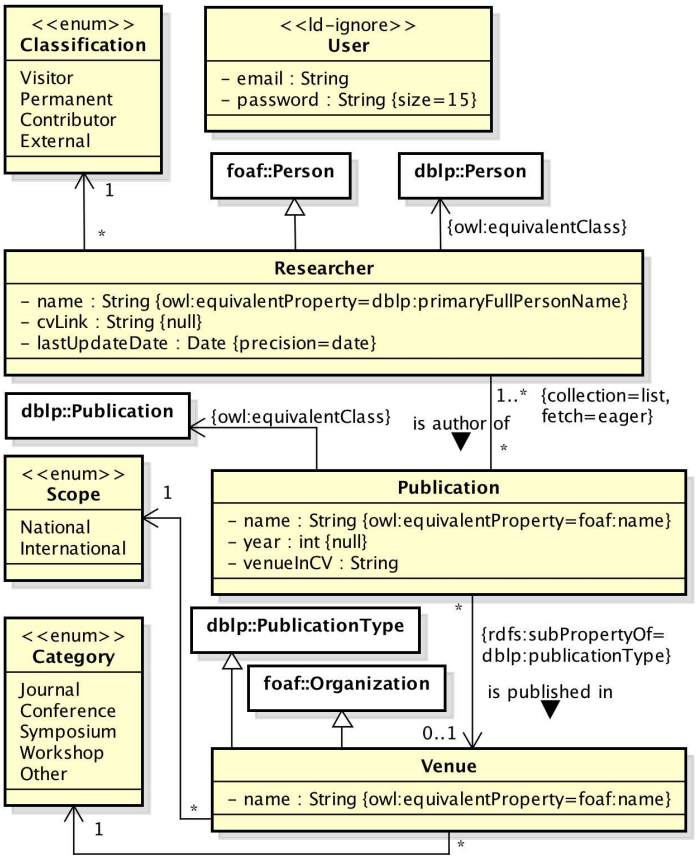

Fig. 13. *FrameWeb-LD* Entity Model with linked data mappings [10].

not least, data from all classes are to be published as linked data, unless the ⟨⟨ld-ignore⟩⟩ stereotype is used (either to exclude specific attributes or entire classes). In the example, the User class is excluded from the linked data set to be published.

Once all the mappings have been included in the Entity Model, tool support[23] can aid developers in producing code for an Ontology-based Data Access (ODBA) solution such as D2RQ, which creates a layer on top of the relational database and offers triple-store features (derreferenceable URIs for navigation, a SPARQL endpoint for querying, etc.) based on a semi-automatic conversion from the database schema to RDF. Listings 3 and 4 show excerpts from the OWL operational ontology and D2RQ mapping generated by *FrameWeb-LD*'s tool support. Some of the mappings of Figure 13 can be identified in these generated artifacts.

[23] The tool is called ReMaT and is available in a stale branch in *FrameWeb*'s source code repository: https://github.com/nemo-ufes/FrameWeb/tree/breno/

Listing 3. Excerpt from operational ontology in OWL generated by *FrameWeb-LD*'s tool support [10].

```
<owl:Class rdf:about="http://dev.nemo.inf.ufes.br/owl/c2d.owl#Publication">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
       Publication</rdfs:label>
  <rdfs:subClassOf rdf:resource="http://dblp.uni-trier.de/rdf/schema
       -2015-01-26#Publication"/>
</owl:Class>
<owl:Class rdf:about="http://dev.nemo.inf.ufes.br/owl/c2d.owl#Venue">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Venue</
       rdfs:label>
  <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1/Organization"/>
  <rdfs:subClassOf rdf:resource="http://dblp.uni-trier.de/rdf/schema
       -2015-01-26#PublicationType"/>
</owl:Class>
<owl:ObjectProperty rdf:about="http://dev.nemo.inf.ufes.br/owl/c2d.owl#
     isPublishedIn">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
       isPublishedIn</rdfs:label>
  <rdfs:domain rdf:resource="http://dev.nemo.inf.ufes.br/owl/c2d.owl#
       Publication"/>
  <rdfs:range rdf:resource="http://dev.nemo.inf.ufes.br/owl/c2d.owl#Venue"/>
  <rdfs:subPropertyOf rdf:resource="http://dblp.uni-trier.de/rdf/schema
       -2015-01-26#publicationType"/>
</owl:ObjectProperty>
```

Listing 4. Excerpt from the relational-to-RDF mapping file generated by D2RQ and *FrameWeb-LD*'s tool support [10].

```
@prefix c2d: <http://dev.nemo.inf.ufes.br/owl/c2d.owl#>

# Table Researcher
map:Researcher a d2rq:ClassMap;
  d2rq:dataStorage map:database;
  d2rq:class c2d:Researcher;
  d2rq:classDefinitionLabel "Researcher";
  rdfs:subClassOf foaf:Person;
  owl:equivalentClass dblp:Person;
  .
map:Researcher_name a d2rq:PropertyBridge;
  d2rq:belongsToClassMap map:Researcher;
  d2rq:property vocab:Researcher_name;
  d2rq:propertyDefinitionLabel "Researcher name";
  owl:equivalentProperty dblp:primaryFullPersonName;
  d2rq:column "Researcher.name";
  .
```

4 Future: where is FrameWeb Going?

FrameWeb is an ongoing research project with undergraduate and graduate students working on different aspects of the proposal. In the previous sections of this paper, we have already mentioned limitations of the approach that need to be addressed in future work. Section 3, for instance, mentioned the support for security frameworks and linked data not being incorporated into the *FrameWeb* Eclipse plug-in, which is on our short-term plans for the future.

Section 2 discussed drawbacks that have motivated recent proposals for the evolution of *FrameWeb*: (i) the language not being generic enough; (ii) not having a precise language specification; (iii) lack of tool support; and (iv) being tailored to a specific

architecture and particular framework categories. As discussed, the evolution of *FrameWeb* only partially addresses these challenges and, thus, there are many opportunities for future work to be taken from these limitations.

Regarding the ***FrameWeb* language generality** and its **precise specification**, the definition of the meta-model and the application of the method in other platforms and with different frameworks have contributed towards these aspects. However, to properly understand each category of framework supported by the method, a more systematic study of the different frameworks of each category is required.

As such, we intend to build ontologies for each supported category, using an ontology engineering approach to try and make sure such reference model properly represents a consensus among the mostly used frameworks. Then, the *FrameWeb* meta-model can be reviewed and adjusted based on the ontology, possibly leading to changes in the *FrameWeb* language.

This is an ongoing effort, conducted under a separate research project that aims at building a *Software Frameworks Ontology Network* (SFWON).[24] The network already includes the *Object/Relational Mapping Ontology* (ORM-O) [31], which was built based on the *Relational Database System Ontology* (RDBS-O) [2] and the *Object-Oriented Code Ontology* (OOC-O) [3], both part of SFWON.

Regarding the **lack of tool support**, the development of the *FrameWeb Editor* and the *FrameWeb Code Generator* is an ongoing effort, with many ideas for new developments, for instance:

- The use of the *FrameWeb* tools in the context of Web Development and Semantic Web courses offered in our university have identified bugs and improvements in the tools' usability, reliability, etc.[25] that need to be fixed so the tools can be put to further use and test;
- The editor currently has no support for the creation of Architecture Definition Files, making it harder for organizations/developers to include support for their platform/frameworks of choice in *FrameWeb*;
- Being based on the Eclipse IDE, the tools integrate best with projects for programming languages and platforms that are supported by this IDE and its plug-ins. Support for different development environments could be offered;
- Modern Rapid Application Development tools (e.g., JHipster[26]) automatically generate considerably more code, especially regarding basic Create, Retrieve, Update and Delete (CRUD) functionalities. These tools, however, are not as extensible (i.e., able to support different frameworks) as *FrameWeb*, but our tools need support for easier generation of CRUD features and other artifacts that are common to WISs;
- Currently, code generation works only in one direction, thus generating code again overwrites any changes that might have been manually performed in previously generated files. Support for preserving manual changes or even more advanced reverse engineering features could be added.

[24] https://nemo.inf.ufes.br/projects/sfwon/
[25] https://github.com/nemo-ufes/FrameWeb/issues
[26] https://www.jhipster.tech

Regarding ***FrameWeb*'s supported architecture and framework categories** further efforts similar to the one presented in Section 3.2 to include support for new categories of frameworks are in order. Such efforts could include the definition of an ontology for the new category of frameworks or postpone the creation of the ontology as a later step (e.g., the support for security frameworks was proposed based on the most used frameworks in the Java platform [24]). As new types of frameworks are included in the method, new architectures can also be proposed.

Finally, we need to perform more extensive and systematic experiments in order to evaluate *FrameWeb* in all of its aspects. Although each proposal conducted their own validation through proofs of concept and small experiments, we do not yet have properly evaluated the entire *FrameWeb* proposal in terms of its usefulness, ease of use, efficacy, etc.

With the *FrameWeb* project, I intend to continue to honor professor Ricardo de Almeida Falbo by carrying on a research agenda that he helped create — more accurately put, would not exist without him — for many years to come. Hopefully, this project will continue to contribute to the qualification of students involved in it, a legacy that Falbo should be proud of.

5 Personal Notes

I met Ricardo as my professor of Requirements Engineering during an undergraduate course in Computer Science at UFES in 2002, the year in which he also became my advisor on a "Scientific Initiation" (undergraduate research scholarship) project. He also supervised my (research-oriented) undergraduate final project in 2004 and accepted me as a Masters student in 2005, continuing to supervise me until 2007. After my PhD abroad, I came back to Brazil and became a professor at the Computer Science Department of UFES in 2013, thus Falbo became my colleague until his retirement in 2019. I am proud to say that during this entire time Ricardo has been, and continues to be, a great friend.

The topic of this paper, the *FrameWeb* method, was born during my Masters course, under the supervision of Falbo [27]. However, the topic of Web Engineering was not connected to my previous undergraduate research projects under his supervision, nor was it one of the particular areas that Ricardo was focusing his research. Instead, it was motivated by my previous experiences in software development projects for the Web and Falbo decided to accept it as the research topic of one of his supervised Masters students. As a professor now myself, I see how altruistic this gesture was at the time and it is fair to say that this contributed to the researcher I came to be, which I think was Ricardo's intention all along (but we will have to ask him).

Professor Ricardo, thank you for all that you have done for me. You are in great part responsible for (as modest as they may be) my academic accomplishments. You are definitely one of my role models and I hope (and work hard) to be to my supervised students as good an advisor as you were to me. Congratulations on a successful career of inspiring people like me.

References

1. OMG: Ontology Definition Metamodel (ODM) Specification, v. 1.1 (formal/14-09-02), http://www.omg.org/spec/ODM/1.1/ (2014)
2. de Aguiar, C.Z., Falbo, R.A., Souza, V.E.S.: Ontological Representation of Relational Databases. In: Proc. of the 11th Seminar on Ontology Research in Brazil (ONTOBRAS 2018). pp. 140–151. CEUR, São Paulo, SP, Brazil (2018)
3. de Aguiar, C.Z., Falbo, R.d.A., Souza, V.E.S.: OOC-O: A Reference Ontology on Object-Oriented Code. In: Proc. of the 38th International Conference on Conceptual Modeling (ER 2019). pp. 13–27. Springer, Salvador, BA, Brazil (2019)
4. de Almeida, N.V., Campos, S.L., Souza, V.E.S.: A Model-Driven Approach for Code Generation forWeb-based Information Systems Built with Frameworks. In: Proc. of the 23rd Brazilian Symposium on Multimedia and the Web (WebMedia 2017). pp. 245–252. ACM, Gramado, RS, Brazil (oct 2017)
5. Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall / Sun Microsystems Press, 2nd edn. (2003)
6. Bauer, C., King, G.: Hibernate in Action. Manning, 1 edn. (2004)
7. Berners-Lee, T.: Linked Data - Design Issues,http://www.w3.org/DesignIssues/LinkedData.html (last access: May 7th, 2015) (2006)
8. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284**(5), 34–43 (2001)
9. Campos, S.L., Souza, V.E.S.: FrameWeb Editor: Uma Ferramenta CASE para suporte ao Método FrameWeb. In: Anais do 16º Workshop de Ferramentas e Aplicações, 23º Simpósio Brasileiro de Sistemas Multimedia e Web (WFA/WebMedia 2017). pp. 199–203. SBC, Gramado, RS, Brazil (oct 2017)
10. Celino, D.R., Reis, L.V., Martins, B.F., Souza, V.E.S.: A Framework-based Approach for the Integration of Web-based Information Systems on the Semantic Web. In: Proc. of the 22nd Brazilian Symposium on Multimedia and the Web. pp. 231–238. ACM (nov 2016)
11. DeMichiel, L., Shannon, B.: JSR 342: Java(TM) Platform, Enterprise Edition 7 (Java EE 7) Specification, https://jcp.org/en/jsr/detail?id=342 (last access: April 29th, 2015) (2013)
12. Ferraiolo, D., Cugini, J., Kuhn, D.R.: Role-based access control (RBAC): Features and motivations. In: Proc. of 11th Annual Computer Security Application Conference. pp. 241–248 (1995)
13. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley, 1 edn. (2002)
14. Fowler, M.: Inversion of Control Containers and the Dependency Injection pattern, http://www.martinfowler.com/articles/injection.html (last access: September 29th, 2016) (2004)
15. Frakes, W.B., Kang, K.: Software reuse research: Status and future. IEEE Transactions on Software Engineering **31**(7), 529–536 (2005)
16. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: No TitleDesign Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1 edn. (1994)
17. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Phd thesis, University of Twente, The Netherlands (2005)
18. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space. Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers, 1 edn. (2011)
19. Ireland, C., Bowers, D., Newton, M., Waugh, K.: A classification of object-relational impedance mismatch. In: 2009 First International Conference on Advances in Databases, Knowledge, and Data Applications. pp. 36–43. IEEE (2009)

20. Martins, B.F.: Uma abordagem dirigida a modelos para o projeto de Sistemas de Informação Web com base no método FrameWeb. Ph.D. thesis, Dissertação de Mestrado, Universidade Federal do Espírito Santo (2016)
21. Martins, B.F., Souza, V.E.S.: A Model-Driven Approach for the Design of Web Information Systems based on Frameworks. In: Proc. of the 21st Brazilian Symposium on Multimedia and the Web. pp. 41–48. ACM (2015)
22. Murugesan, S., Deshpande, Y., Hansen, S., Ginige, A.: Web Engineering: a New Discipline for Development of Web-Based Systems. In: Murugesan, S., Deshpande, Y. (eds.) Web Engineering - Managing Diversity and Complexity of Web Application Development, chap. 1, pp. 3–13. Springer (2001)
23. Pastor, O., España, S., Panach, J.I., Aquino, N.: Model-driven development. Informatik-Spektrum **31**(5), 394–407 (2008)
24. do Prado, R.C., Souza, V.E.S.: Securing FrameWeb: Supporting Role-based Access Control in a Framework-based Design Method for Web Engineering. In: Proc. of the 24th Brazilian Symposium on Multimedia and the Web (WebMedia '18). pp. 213–220. ACM, Salvador, BA, Brazil (2018)
25. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects. Wiley (2013)
26. Silva, L.R.M.: Integração do Editor de Modelos de FrameWeb à IDE Eclipse. Tech. rep., Relatório Final de Pesquisa, Programa Institucional de Iniciação Científica, Universidade Federal do Espírito Santo (2019)
27. Souza, V.E.S.: FrameWeb: um Método baseado em Frameworks para o Projeto de Sistemas de Informação Web. Tech. rep., Universidade Federal do Espírito Santo (2007)
28. Souza, V.E.S., Falbo, R.A., Guizzardi, G.: Designing Web Information Systems for a Framework-based Construction. In: Halpin, T., Proper, E., Krogstie, J. (eds.) Innovations in Information Systems Modeling: Methods and Best Practices, chap. 11, pp. 203–237. IGI Global, 1 edn. (2009)
29. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF - Eclipse Modeling Framework. Addison-Wesley, 2nd editio edn. (2008)
30. Viyovic, V., Maksimovic, M., Perisic, B.: Sirius: A rapid development of DSM graphical editor. In: Intelligent Engineering Systems (INES), 2014 18th International Conference on. pp. 233–238. IEEE (2014)
31. Zanetti, F.L., de Aguiar, C.Z., Souza, V.E.S.: Representação Ontológica de Frameworks de Mapeamento Objeto/Relacional. In: Proc. of the 12th Seminar on Ontology Research in Brazil (ONTOBRAS 2019). CEUR, Porto Alegre, RS, Brasil (2019)
32. Zupeli, B.L., Souza, V.E.S.: Integração de um Gerador de Código ao FrameWeb Editor. In: Anais Estendidos do 24º Simpósio Brasileiro de Sistemas Multimedia e Web - Workshop de Ferramentas e Aplicações (WFA/WebMedia 2018). pp. 109–113. SBC, Salvador, BA, Brazil (2018)