

A UML Profile for Modeling Framework-based Web Information Systems

Vítor Estêvão Silva Souza¹, Ricardo de Almeida Falbo¹, Giancarlo Guizzardi^{1,2}

¹ Universidade Federal do Espírito Santo, Av. Fernando Ferrari, 514
29075-910 Vitória – ES, Brazil

² Laboratory for Applied Ontology, Polo Tecnologico, Via Solteri, 38
38100 Trento, Italy
vitorsouza@gmail.com, falbo@inf.ufes.br, guizzardi@loa-cnr.it

Abstract. The rapid evolution of the area of Web Engineering has motivated the proposal of several methods and frameworks for the development of Web Information Systems (WISs). In particular, it is becoming more and more common to use container-based architectures and frameworks when it comes to their development. Following this idea, we have proposed a method for designing framework-based WISs, called FrameWeb. and, in this paper, we present FrameWeb's UML profile for modeling framework components in design models.

Keywords: Web Engineering, Web Information Systems, Frameworks, Modeling Language, UML profile.

1 Introduction

Web Engineering (WebE) can be defined as “the establishment and use of engineering principles and disciplined approaches to the development, deployment and maintenance of Web-based Applications” [1]. What started with the ad-hoc development of simple web pages now congregates many methods for the design of Web Information Systems (WISs), such as WAE [2] and OOWS [3].

Also, new technologies for coding WISs have rapidly evolved. Frameworks for the solution of infrastructure issues were developed and entire architectures providing containers for user components became standards. For example, the most recent version of Java EE [4] borrows many concepts from existing frameworks. These technologies provide a large code base that can be reused, improving productivity.

This scenario motivated the proposition of FrameWeb [5]: a Framework-Based Design Method for Web Engineering. FrameWeb proposes a basic architecture for developing WISs and a UML profile for a set of design models that brings concepts used by some categories of frameworks. The use of FrameWeb can further improve team productivity by allowing designers to produce diagrams that represent

framework concepts, and developers (maybe, in the future, CASE tools) to quickly and directly translate these diagrams to code.

The objective of this paper is to present FrameWeb's modeling profile. It is organized as follows: section 2 talks about the area of Web Engineering and what has motivated the authors for the creation of this UML profile. This section also summarizes the key points of FrameWeb's design method. Section 3 details FrameWeb's modeling profile, which is the focus of this paper; section 4 compares FrameWeb's profile to others; finally, section 5 concludes and lists future work.

2 Web Engineering

Web sites evolved from static pages to complex information systems. In this process, many frameworks (artifacts of code that provide components ready for reuse by inheritance, composition or configuration) were created to help Web developers speed up the development of Web applications.

These frameworks substantially change the kind of components that are built during coding. Classes that inherit from other classes in the framework, configuration files and others are now source code artifacts that must be written by developers. In particular, three kinds of frameworks have the most impact on this change: Front Controller, Object/Relational Mapping and Dependency Injection frameworks.

Front Controller [6] frameworks, also known as MVC frameworks for being based on the well-known Model-View-Controller architectural pattern [7], separate the functionality of a WebApp from its presentation. With proper configuration, the developer is given the task of creating action classes that will respond to client requests and redirect the response to a user interface (UI) component. In FrameWeb, action classes, UI components and their interaction (which guides the configuration of the framework) are modeled in the Navigation Model, presented in section 3.3.

Object/Relational Mapping (ORM) frameworks provide automatic and transparent persistence of objects to tables of a Relational Database, using meta-data that describe the mapping between both worlds [8]. A mapping is written for each persistent class and the ORM framework does the rest of the job. In FrameWeb, these mappings are presented in the Domain Model (section 3.1), while the classes that perform the persistence are modeled in the Persistence Model (section 3.2).

Dependency Injection (DI) frameworks allow the developer to program to interfaces [9, 7] and specify the concrete dependencies in a configuration file. The idea is that classes that depend on services from different tiers would declare an association with an interface instead of the concrete implementation. This facilitates, for instance, the replacement of the real service class with a mock object for unit testing. When an object is created, the DI framework wires all required dependencies automatically, based on a configuration file. In FrameWeb, the dependency relations between different tiers are represented in the Application Model (section 3.4).

A detailed description of these and other kinds of frameworks can be found at [5].

Our personal experience with the use of frameworks for the construction of WISs has motivated the proposal of the Framework-Based Design Method for Web Engineering (FrameWeb) [5]. The method has two main propositions:

- ◆ A standard architecture for WISs that integrates with the frameworks described earlier by separating their concerns into different packages;
- ◆ A UML profile suited for the construction of four kinds of design models that represent framework components from different packages: Domain Model, Persistence Model, Navigation Model and Application Model.

FrameWeb's standard architecture separates different frameworks concerns into three layers. At the Presentation layer (**View** and **Controller** packages), UI components and action classes integrate with the Front Controller framework, calling services from a Business layer, which contains service classes (**Application** package) and domain classes (**Domain** package). The latter, in turn, are mapped by the ORM framework and persisted by classes that belong to the Data Access layer (**Persistence** package). Class dependencies that cross a layer's boundaries are taken care of by the DI framework.

For a detailed description of the method, the interested reader should refer to [5]. In the next section, we focus on FrameWeb's modeling profile, used during design to produce diagrams that depict coding artifacts that belong to these different layers.

3 FrameWeb's Modeling Profile

To model classes and other components, FrameWeb defines a UML profile using standard extension mechanisms for designing four different kinds of diagrams during system design: Domain Model, Persistence Model, Navigation Model and Application Model. All of them are based on UML's class diagram, but represent components from different layers that integrate with different frameworks.

Throughout the next subsections we detail these four models using as examples diagrams produced for a collaborative learning environment called JSchool¹, developed by the local Java User Group² using FrameWeb. JSchool contains a core subsystem, which is responsible for user registration, login, system configuration and allowing users to send messages to the administrators and view other users' profiles. JSchool also includes a group module, which contains features for creating, managing, searching, joining and participating in groups.

Figure 1 shows the conceptual model for the core and group modules. Collaborators are registered with name, e-mail, password, institution, contact information and picture. Groups have names, descriptions, keywords and configurations such as if the group is open or public. Other classes complement these two, representing invitations to participate in a group, confirmation of a user's e-mail address, a user membership and role in a group and requests from a user to create a group or participate in a group.

¹ <http://jschool.dev.java.net>

² Espirito Santo Java User Group (ESJUG) – <http://esjug.dev.java.net>

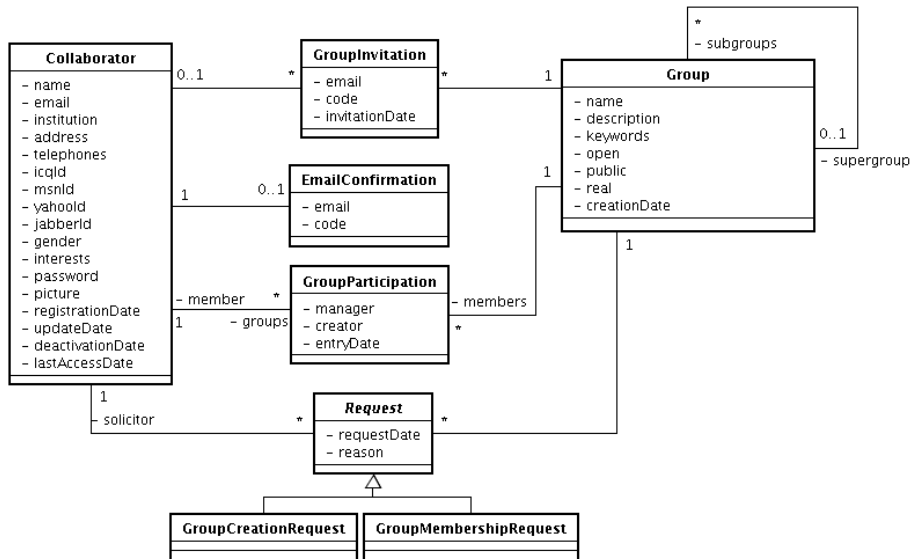


Fig. 1. Conceptual model for the core and group modules of JSchool.

3.1 Domain Model

The Domain Model is a UML class diagram that represents classes of objects from the problem domain and their Object/Relational (OR) mappings. This diagram should be used by developers to guide the codification of classes from the **Domain** package.

Designers should use the conceptual model built during Requirement Analysis and add platform-specific details and the OR mappings. Platform-specific details include attribute types, association navigabilities, etc. The OR mappings will guide the developers on how to configure the ORM framework to persist these entities.

Although this configuration is more related to persistence, it is represented in the Domain Model because the classes that should be mapped and their attributes are represented in this diagram.

Table 1 describes some possible OR mappings for the Domain Model. For each mapping, the table presents which lightweight extension mechanism was used and what are the possible values or syntax for its use. None of the mappings are mandatory and most of them have sensible defaults in order to lessen the amount of information on the model. These default values are shown in the possible values column in boldface or in parenthesis.

The Domain Model for the core and group modules of JSchool is shown in figure 2. According to the default values, all classes are persistent and class and attribute names are used as table and column names respectively. None of the classes have ID or version attributes because they are inherited from a utility package.

Attributes have received mappings such as nullability and size. Date attributes such as **invitationDate**, **entryDate** and **requestDate** were mapped as date-only precision, while the remaining dates will be stored as timestamps. All three collections (navigable association endings with multiplicity “many”) were configured

to be sorted naturally (will be implemented in the programming language, Java), to be fetched lazily (only when used for the first time) and to cascade all operations (e.g. if a group is deleted, all of its subgroups are automatically deleted).

Table 1. Possible OR mappings for the Domain Model.

Mapping	Extension	Possible Values
If the class is persistent, transient or mapped (not persistent itself, but its properties are persistent if another class inherits them)	Class stereotype	<< persistent >> <<transient>> <<mapped>>
Name of the table in which objects of a class will be persisted	Class constraint	table= <i>name</i> (default: class' name)
If an attribute is persistent or transient	Attribute stereotype	<< persistent >> <<transient>>
If an attribute can be null when the object is persisted	Attribute constraint	null not null
Date/time precision: store only the date, only the time or both (timestamp)	Attribute constraint	precision = (date time timestamp)
If the attribute is the primary-key of the table	Attribute stereotype	<<id>>
How the ID attribute should be generated: automatically, obtained in a table, use of IDENTITY column, use of SEQUENCE column or none	Attribute constraint	generation = (auto table identity sequence none)
Name of the column in which an attribute will be persisted	Attribute constraint	column= <i>name</i> (defaults to the attribute's name)
Size of the column in which an attribute will be persisted	Attribute constraint	size= <i>value</i>
Inheritance mapping strategy: one table for each class using UNION, one table for each class using JOIN or single table for the entire hierarchy	Inheritance stereotype	<<union>> <<join>> << single-table >>
Type of collection which implements the association: bag, list, set or map	Association constraint	collection = (bag list set map)
Order of an association's collection: natural ordering (implemented in code) or order by columns (ascending or descending)	Association constraint	order = (natural <i>column names</i> [asc desc])
Cascading of operations through the association: nothing, persists, merges, deletions, refreshes or all	Association constraint	cascade = (none persist merge remove refresh all)
Association fetching strategy: lazy or eager.	Association constraint	fetch = (lazy eager)

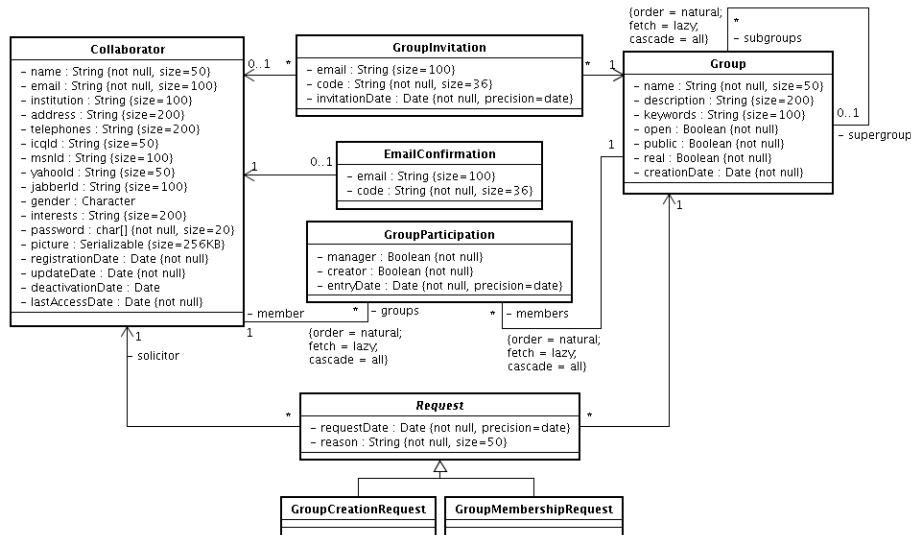


Fig. 2. Domain Model for the core and group modules of JSchool.

3.2 Persistence Model

FrameWeb indicates the use of the Data Access Object (DAO) pattern [6] for the construction of the Data Access layer. The Persistence Model is a UML class diagram that represents the DAO classes responsible for the persistence of domain objects. This diagram is used by developers for codifying classes from the **Persistence** package.

For each domain class that needs to be persisted, the Persistence Model should present one DAO interface and one or more DAO classes, meaning that different persistence technologies can be implemented in the same system. Base DAO interface and implementation can be presented to reduce the amount of diagram elements in the Model: all DAO interfaces/classes automatically extend the base DAO interface/class, without the need to explicitly show it. Implementing common methods such as **save()**, **delete()** and **retrieveById()** in the base DAO can reduce the amount of modeling that has to be done.

Besides those basic methods, DAOs should also have methods to retrieve objects using specific queries. Also, it is assumed that the interface declares all methods presented by the DAO class to relief the designer of repeating them on the diagram. Thus, basically the Persistence Model represents the “query methods” and does not require any UML extension for such a task. For instance, to authenticate a collaborator's password we need to retrieve the collaborator object given his/her email. Therefore, the method **retrieveByEmail(email: String) : Collaborator** is modeled for the **CollaboratorDAO** implementation.

3.3 Navigation Model

The Navigation Model is a UML class diagram that represents the different components that form the Presentation Logic tier, such as Web pages, HTML forms and action classes. Table 2 shows the stereotypes used by the different components that can be represented in a Navigation Model. This model is used by developers to guide the codification of classes from the **View** and **Controller** packages.

Table 2. Stereotypes for classes in the Navigational Model

Stereotype	What it represents
none	An action class. Class to which the Front Controller framework delegates the action execution.
<<page>>	A static or dynamic Web page.
<<template>>	A template of a Web page. The templates are processed by a template engine that returns an HTML output for the browser.
<<form>>	A HTML input form.
<<binary>>	Any binary file that could be downloaded (an image, a report, a file, etc.).

Dependency associations among these components dictate the control flow when an action is executed. When directed toward an action class, it means that data is coming from pages or forms and are being set at the action class. Analogously, when directed outward an action class, it means that data from that class is being displayed in some page or template. Homonymous attributes in pages, forms and action classes indicate the information being exchanged.

The designer is free to chose the granularity of the action class: one for each use case scenario, one for each use case, and so on. Since an action class can implement many actions with different methods and each of them can produce different outcomes, constraints at the dependency associations indicate which method/result is being modeled, when needed.

Figure 3 shows the Navigation Model for the “Send Message to Administrators” use case. The **web::index** page and **web::home** template represent the pages to which visitors and registered users, respectively, have access. These pages can request the action's input (a form to be filled and submitted), which results in the displaying of the **sendMessageAdministration** template. This template contains an input form that sends the name, email and body of message to the action that, after executed, displays the **messageSent** template, if successful. When a registered user requests the input form, the action retrieves its name and e-mail and automatically fills those input fields. That is why the template has **name** and **email** as attributes.

At the time FrameWeb was being conceived, a question arose as to whether UML's sequence diagram would be better suited for the Navigation Model than the class diagram. The issue was discussed with the first group of developers which used FrameWeb in its early stages to build a WIS. Two main reasons contributed to the choice of the class diagram: (a) it provides a better visualization of the inner structure

of action classes, forms and templates; and (b) it models page – form composition with a more appropriate notation (UML's composition association). In spite of this, FrameWeb does not suggest sequence diagrams should be avoided, but simply recognizes that class diagrams are better suited for the Navigation Model.

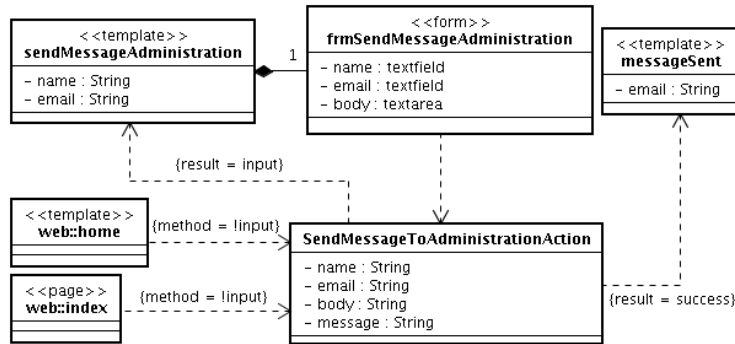


Fig. 3. Navigation Model for the use case “Send Message to Administrators” (JSchool core)

3.4 Application Model

The Application Model is a UML class diagram that represents service classes, which are responsible for the implementation of use cases. This diagram is used for guiding the codification of classes from the **Application** package and the configuration of the dependencies between **Controller**, **Application** and **Persistence** packages.

As with action classes, the designer should chose the granularity of the service classes and display, for each of them, one interface and one implementation. The Application Model displays, then, all action classes and their dependencies with the service interface, the methods of the service class that implement the use case scenarios and DAO interfaces and the dependencies that service classes have on them.

Figure 4 shows part of the Application Model of JSchool's core module, showing the service class for the “Configure System” use case and its dependencies. The action class (tagged with the **controller::** namespace) depends on the service class for the execution of the use case. The service class, in turn, depends on the DAO classes for **Collaborator** and **Group**, as it needs to access data of both classes.

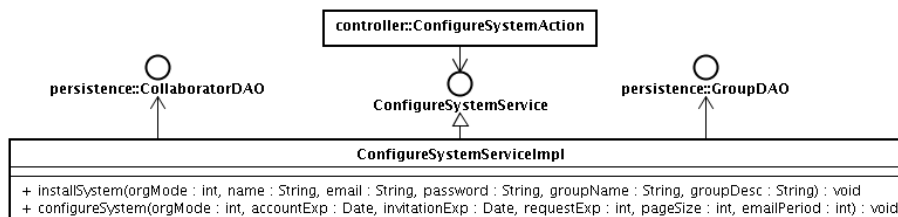


Fig. 4. Part of an Application Model of the core module of JSchool.

4 Related Work

There are several Web Engineering methods that define modeling languages and profiles for Web-specific purposes.

Conallen [2] proposes a UML profile named Web Application Extensions (WAE). WAE extends UML to provide Web-specific constructs for modeling WISs, including a new model called User Experience (UX) Model, which defines guidelines for modeling layout and navigation. Models like the navigation diagram, the class diagram and the component diagram (the last two specific for the web tier) use WAE to represent Web components such as screens, server pages, client pages, forms, links and many more. FrameWeb's modeling profile is quite similar to WAE, as both extend UML for the creation of diagrams that represent web-related elements. However, as FrameWeb is based on frameworks, its stereotypes and constraints are different from those proposed by WAE. Also, we prefer to use dependency associations to represent the relations among web components instead of regular associations, which represent, in other contexts, a direct relation between components and can confuse developers accustomed to this other meaning. This explains why FrameWeb is not an extension of WAE, but a new profile altogether.

OOWS (Object Oriented Web Solution) [3] uses UML for most of its models, making use of its extension mechanisms. But it also proposes extensions that are not standard, which can make things difficult for developers that do not have CASE tools specifically designed for the method. On the other hand, its navigation model defines specific indexing and filtering mechanisms that make it easier to model these kinds of structures, which are quite common in the Web environment.

The UML-based Web Engineering (UWE) [10] also defines a UML profile specifically for Web components using standard extensions mechanism, but also offering a few non-standard extensions. As with WAE, it doesn't define specific extensions for framework-related components either.

FrameWeb uses only lightweight extension mechanisms, being easily supported by most CASE tools that support UML.

Other proposals, such as WebML [11], define a modeling language that is not UML-based. As intuitive as their graphical representations might be, not being based on UML is a big disadvantage for reasons of developer's acceptance and tool support.

Given all of the options available, FrameWeb comes in as a good choice when it comes to architectures that are based on the use of frameworks. In this case, which is very common for the development of WISs, FrameWeb's profile provides models that are directed towards the frameworks' architectures, allowing for quick understanding of the implementation. During two software projects developed to informally experiment the method and its modeling profile, developer feedback indicated that FrameWeb is easy to learn and use, noting some difficulty only on Navigation Models.

5 Conclusions and Future Work

This paper presented a detailed view of the UML profile proposed by FrameWeb, a design method suited for the development of framework-based Web Information Systems. The profile aims at depicting framework-related components, easing the transition from design models to coding. Using lightweight extensions of UML makes the language more familiar to UML developers.

Future work has already started on the experimentation of the method with many different frameworks, to verify if different framework implementations also fit in the categories summarized in section 2. Other opportunities lie in developing a CASE tool to produce FrameWeb's design models and developing automatic code generation tools (feature that is present in other approaches, such as OOWS and UWE).

Although two software projects have already been conducted to informally evaluate the method [5] and modeling profile, systematic evaluations could be proposed to reach conclusions that have more scientific value.

Acknowledgments. This work was accomplished with the financial support of CAPES, an entity of the Brazilian Government reverted to scientific and technological development.

References

1. Murugesan, S., Deshpande, Y., Hansen, S., Ginige, A.: Web Engineering: A New Discipline for Development of Web-based Systems. Proceedings. of. the First ICSE Workshop on Web Engineering. IEEE, Australia (1999)
2. Conallen, J.: Building Web Applications with UML. 2nd edn. Addison-Wesley (2002).
3. Fons, J., Valderas, P., Ruiz, M., Rojas, G., Pastor, O.: OOWS: A Method to Develop Web Applications from Web-Oriented Conceptual Models. Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics. Orlando, FL – USA (2003)
4. Shannon, B.: Java™ Platform, Enterprise Edition (Java EE) Specification, v5. Sun Microsystems (2006)
5. Souza, V. E. S., Falbo, R. A.: FrameWeb: A Framework-based Design Method for Web Engineering. Proceedings of the Euro American Conference on Telematics and Information Systems (EATIS 2007). Faro, Portugal (2007)
6. Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall / Sun Microsystems Press (2001)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994)
8. Bauer, C., King, G.: Hibernate in Action. 1st edn. Manning (2004)
9. Schmidt, D.: Programming Principles in Java: Architectures and Interfaces (<http://www.cis.ksu.edu/~schmidt/CIS200/>). Chapter 9. Capture on February 14th (2007)
10. Koch, N., Baumeister, H., Hennicker, R., Mandel, L.: Extending UML to Model Navigation and Presentation in Web Applications. Proceedings of Modelling Web Applications in the UML Workshop (2000)
11. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language: a modeling language for designing Web sites. Computer Networks. Elsevier (2000) v. 33, n. 1-6, p. 137-157