FrameWeb: A Framework-based Design Method for Web Engineering

Vítor Estêvão Silva Souza, Ricardo de Almeida Falbo
Computer Science Department, Federal University of Espírito Santo, Brazil
vitorsouza@gmail.com, falbo@inf.ufes.br

Abstract

Web application development has long evolved from CGI scripting in structured programming languages to a whole new discipline called Web Engineering. Today, large and complex distributed systems are being built for the web, mostly with the use of frameworks or a container-based architecture. This paper proposes a method for the design of Web applications based on the use of frameworks, including a modeling language that extends UML to build diagrams that specifically depict framework-related components.

Keywords

Web Engineering, Web Information System, Frameworks, UML Profile, Design Method

1. Introduction

First-generation Web Applications (WebApps) were constructed in an ad-hoc manner, without a methodology or a software process to support the development team. While this may work for simple applications, it is inconceivable nowadays to have a WebApp developed with no concern for Software Engineering principles.

Thus, a new discipline and research field was born. Web Engineering (WebE) is the establishment and use of engineering principles and disciplined approaches to the development, deployment and maintenance of Web-based Applications [1].

In this new field of research, a lot of methods, frameworks and modeling languages have been proposed. Conte et al. reference quite a few in [2], but we can also cite WebML [3], WAE [4], OOWS [5], UWE [6], OOHDM [7] and many others.

Along with these researches, technologies for codifying WebApps have also evolved. The use of frameworks or container-based architectures to provide a solid Web infrastructure for the application to be built upon is state-of-the-practice. This Web infrastructure commonly includes a Model-View-Controller (MVC) [8] (or Front Controller [9]) architecture, a dependency injection mechanism [10], automatic object/relational mapping for persistence [11] and more. The use of these frameworks speeds up the coding phase by reusing code that has already been coded, tested and documented by 3rd parties.

These frameworks motivated the Software Engineering Lab (LabES) of the Federal University of Espírito Santo (UFES) to develop a WebE design method that focuses on them. This work started in [12] and now is consolidated in the Framework-based Design Method for Web Engineering (FrameWeb). FrameWeb proposes a basic architecture for developing WebApps and an UML profile for a set of design models that brings concepts used by some categories of frameworks.

This paper presents FrameWeb and is organized as follows: section 2 discusses some issues concerning WebE, with a special focus on existing frameworks that are commonly used; section 3 presents FrameWeb using a real example to illustrate it; section 4 talks about related works and how FrameWeb compares to them. Finally, section 5 presents our conclusions and future work.

2. Web Engineering

First generation Web applications (WebApps) were usually developed in an ad hoc manner, with no concern for Software Engineering principles. Today, however, it is common sense that this kind of applications has to be built around a Software Engineering apparatus compatible with its size and non-functional requirements. To deal with these characteristics, Web Engineering (WebE) was born.

Web Engineering can be defined as "the establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based systems and applications" [1].

As with conventional software engineering, a WebE process starts with the identification of the business needs, followed by project planning. Next, requirements are detailed and modeled taking into account the analysis and design perspective. Then the application is built using tools specialized for the Web. Finally, the system is tested and delivered to end-users [13].

Considering that the platform to which we will develop a software possibly is not taken into account before the design phase of the software development process, developing a WebApp would be just like developing any other application up to that phase. However, many differences between Web Engineering and Conventional Software Engineering have been identified by researchers and practitioners [14], among which we highlight the

short time frames for delivering the application [13]. This urgency influences the software process as a whole, suggesting that faster approaches should be preferred.
Looking for agility, especially in the coding phase, several frameworks have been developed, especially if we consider popular technologies, such as Java, .NET and PHP. In this context, a framework is seen as an artifact code that provides components ready for reuse by inheritance, composition or configuration. When combined, these frameworks allow large WebApps to be built with n-tier architectures, relieving the programmers of a lot of coding effort and making the development faster and more productive.

2.1. Frameworks for Web development

In our experience with the development of WebApps using the Java platform, we learned about and worked with quite a few frameworks. Container-based architectures (e.g. Java Enterprise Edition) have adopted many concepts from successful frameworks so we can consider them as sets of frameworks brought together.
The use of these frameworks has a considerable impact in the architecture of a WebApp. Since it's possible to find many frameworks for the exact same task, we categorized them by objectives:

2.1.1. MVC frameworks. MVC stands for Model-View-Controller [8] and is a software architecture that has found great acceptance by Web developers. When applied to the Web, the MVC architecture is adapted and receives a new name: "Front Controller" [9]. Both terms are used indistinguishably by Web developers.
When structured using the MVC architecture, a WebApp manages all requests from clients using an object known as Front Controller. This object decides which class will respond to the current request (the action class). Then, it instantiates an object of that class and delegates the control to it, expecting some kind of response afterwards. Based on that response, the controller decides the appropriate view to present as result, such as a web page, a template, a report, a file download, among other possibilities.
MVC Frameworks usually provide the front controller, a superclass or interface for action classes, several result types and a well defined syntax for configuration files. There are more than 50 MVC frameworks for the Java platform alone. Some of the most popular are Struts[1] and WebWork[2].

2.1.2. Decorator frameworks. Decorator frameworks automate the otherwise tedious task of making every web page of the site have the same layout (header, footer, navigation bar, colors, images, etc).
They work like the Decorator design pattern [8], providing a class that intercepts requests and wraps their responses with an appropriate layout before it is returned to the client. It also provides dynamic selection of decorators, making it easy to create alternate layouts, such as a "print version". Examples are Tiles[3] and SiteMesh[4].

2.1.3. Object/Relational Mapping frameworks. Relational Database Management Systems (RDBMS) have long been the *de facto* standard for data storage. Even object oriented application still use it for object persistence, giving rise to a "paradigm mismatch" [11].
Among the many options to deal with this problem, there is the Object/Relational Mapping (ORM) approach, which is the automatic and transparent persistence of objects to tables of a RDBMS using meta-data that describes the mapping between both worlds [11]. The use of ORM frameworks is not restricted to Web applications and has been in use for quite some time now in all kinds of software. The most popular ORM framework is Hibernate[5]. Other well-known frameworks are Java Data Objects[6] and Apache Object Relational Bridge[7].

2.1.4. Dependency Injection frameworks. Object-oriented applications are usually built in tiers, each of which having a separate responsibility. According to [10], when we create classes that depend on objects of other classes to perform a certain task, it is preferred that the dependent class is related only to the interface of its dependencies, and not to a specific implementation of that service. This is a good practice in programming known as "programming to interfaces, not implementations" [16, 8].
Dependency Injection (DI) frameworks allows the developer to program to interfaces and specify the concrete dependencies in a configuration file. When a certain object is obtained from the DI framework, all of its dependencies are automatically injected and satisfied.
As well as ORM frameworks, DI frameworks aren't exclusively for WebApps, although they tend to integrate more seamlessly with applications that run inside containers, just like a WebApp runs inside a Web server. Lots of frameworks provide this service, including Spring Framework[8] and PicoContainer[9].

2.1.5. Other frameworks. Other kinds of frameworks can also take part on the development of a WebApp, including AOP frameworks, Authentication and Authorization frameworks, search engines, cache solutions and many others.
In spite of frameworks being much used, there is no Web Engineering method that explores their use in the design phase of the software process. To fill this gap, we propose FrameWeb, a Framework-based Design Method for Web Engineering, which is presented in the next section.

1 http://struts.apache.org
2 http://www.opensymphony.com/webwork
3 http://struts.apache.org/struts-tiles
4 http://www.opensymphony.com/sitemesh
5 http://www.hibernate.org
6 http://java.sun.com/products/jdo
7 http://db.apache.org/ojb/
8 http://www.springframework.org
9 http://www.picocontainer.org

3. FrameWeb

FrameWeb is a method for designing WebApps that assumes the use of certain kinds of frameworks during the software process. It defines a basic architecture for WebApps and proposes design models that are closer to their implementation using those frameworks.

FrameWeb does not prescribe a rigid software process, but it does expect that use case and class diagrams are developed during requirement analysis. Also, as mentioned earlier, one of the motivations for the creation of FrameWeb is the demand for agility that surrounds WebApp projects. Thus, although the method brings more agility especially to the design and coding phases, developers are advised to follow principles of agility during requirements analysis, as the ones proposed by Agile Modeling [15].

Design is where the method focuses most on its propositions. Since the implementation platform is considered, a standard architecture and a set of models specifically tailored for framework-based applications are used during this phase.

The coding phase is greatly facilitated by the use of frameworks, especially because design models already show framework-related components and their relationship among themselves. The use of frameworks can also have an impact on testing and deployment, but FrameWeb does not make any considerations about these final stages.

To illustrate the use of the method, we partially present throughout the rest of this paper its application to the development of a WebApp, the LabES Portal. Figure 1 shows its use case diagram, simplified for brevity.

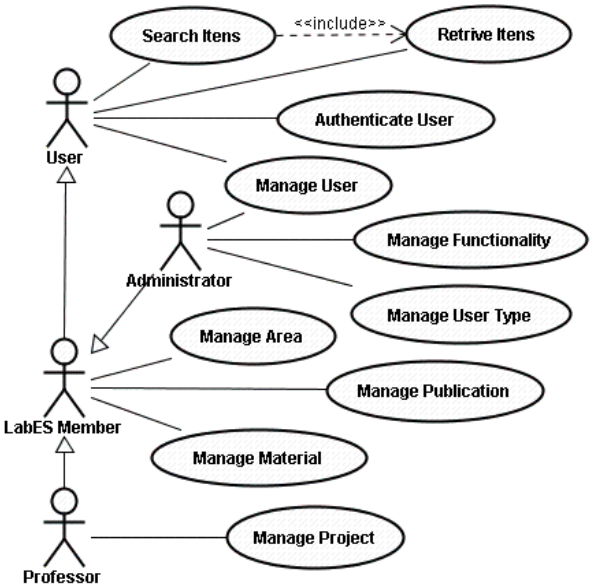

Figure 1. A simplified use case diagram for LabES Portal.

To provide a better interaction with the Software Engineering community, the Software Engineering Lab (LabES) of the Federal University of Espírito Santo developed a website named "LabES Portal", which was developed using FrameWeb. This WebApp has a basic set of services providing information about current LabES projects, areas of interest, publications and other material available for download, among others.

During analysis, class diagrams were constructed to represent the concepts involved in the problem domain, without taking into account implementation aspects.

Since FrameWeb focus on the design phase, in this paper we discuss it with more details. As previously mentioned FrameWeb proposes: (i) a standard software architecture that structures the system into layers that integrate well with the framework categories presented in section 2, and (ii) a set of design models that brings concepts used by these framework categories using an UML profile developed to make these diagrams closer to their implementations. These two propositions are further detailed in the subsections that follow.

3.1. Framework-based WebApp Architecture

Considering the framework categories presented previously, FrameWeb proposes the use of the standard software architecture for WebApps shown in Figure 2. This architecture organizes the system in three tiers: presentation, business and data access logic.

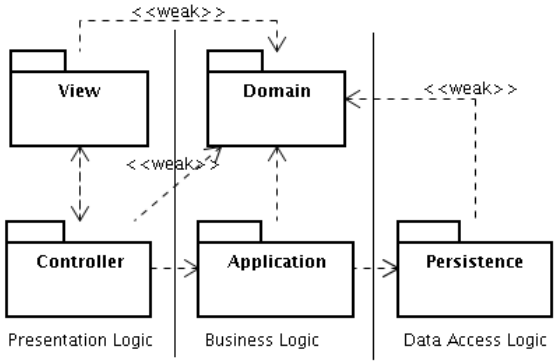

Figure 2. FrameWeb's standard architecture for WebApps

The first tier concerns the graphical user interfaces. The `View` package contains web pages, style sheets, images, layout templates and other files related exclusively with the exhibition of information to the user. The `Controller` package comprises action classes and other files related to the MVC framework. Both packages depend on each other: view elements send user inputs to controller classes that, in turn, process responses using view elements again.

The second tier is where the business logic is implemented. It is divided in two packages: the `Domain` package includes the business domain concepts identified

and modeled by class diagrams during analysis and refined during design; the `Application` package implements the use cases defined in the requirement specification. As the latter manipulates objects of the former, it has a dependency association to it.
The `Controller` package from the Presentation tier depends on the `Application` package, because it mediates the user access to the functionalities provided by the software. User input coming from the `View` are translated into use case execution (methods of `Application` classes) by the action classes (`Controller` classes).
Both `Controller` and `View` have a dependency to the `Domain` package with the stereotype `<<weak>>`. This stereotype indicates loose coupling, since they use domain objects only to display their data (`View`) or to pass them around as parameters (`View` / `Controller`).
The third and last tier (the `Persistence` package) is responsible for storing persistent objects in long-term duration media, such as databases. Along with the use of an ORM framework, FrameWeb suggests the use of the Data Access Object (DAO) design pattern [9]. The DAO pattern adds an extra abstraction layer, decoupling the data access logic layer from the persistence technology, allowing developers to change to another ORM framework (or even to an OO database), if needed.
The `Application` package from the business tier depends on the `Persistence` package to persist objects as a result of an use case execution. Persistence manipulates `Domain` objects and, thus, this weak dependency association is also represented.
This architecture provides a solid base for WebApps that make use of the frameworks presented in subsection 2.1. Each package contains classes that integrate with these frameworks. To model these classes, an UML-based modeling language is proposed by FrameWeb, which is presented in the next subsection.

3.2. Modeling Language

To model classes that integrate with the frameworks, we felt the need for a specific modeling language that would represent concepts that come from the use of these frameworks.
Following the same approach as other modeling languages, such as WAE [4] and UWE [6], FrameWeb defines extensions to the UML meta-model to model typical web and framework-related components, creating an UML profile for use in four diagrams: domain model, persistence model, navigation model and application model. Each of these diagrams and the extensions associated to them are explained in the subsections that follow.

3.2.1. Domain Model. The domain model is an UML class diagram that represents objects from the problem domain, which later will guide the implementation of the `Domain` package. Additionally, some information about their mapping to relational data bases is added, in order to guide the configuration of the ORM framework that will be used for persistence.
O/R mappings aside, building the domain model is just like building a regular domain design class diagram. Since its classes were first modeled during analysis, you start from them and add information that is platform-dependent (usually attribute types and association navigabilities). .
Concerning the O/R mappings, the domain model also introduces information that models the meta-data needed by the ORM framework to transparently persist objects, providing information such as if a given class is persistent or transient, in which table the class is going to be persisted, if a given attribute can be null or not, among others. Most mapping options have sensible defaults in order to reduce the amount of work to be done. Table 1 shows some of the proposed mappings. The first column shows the information being mapped; the second column indicates which UML extension mechanism is used for the mapping; the third column lists possible values and the format in which they are to be written; the fourth and last column presents the value to be considered if none is specified (default value).

Table 1. Possible mappings for the domain model.

	UML Extension	***Possible Values***	***Default Value***
If the class is persistent, transient or mapped	Class stereotype	persistent transient mapped	persistent
If an attribute is persistent or transient	Attribute stereotype	persistent transient	persistent
Date/time precision	Attribute stereotype	precision= (date | time | timestamp)	precision = timestamp
If an attribute is the primary key	Attribute stereotype	id	–
Size of an attribute	Attribute constraint	size=*value*	–
Collection ordering	Association constraint	order= (natural | *column*)	–

Figure 3 shows part of the domain model for the LabES Portal. According to the mappings, by default, all classes are persistent and all attributes are nullable, except those marked with a `{not null}` constraint. The precision for the `birthDate` and `availabilityDate` attributes was set to date only. The self-association in `Area` is

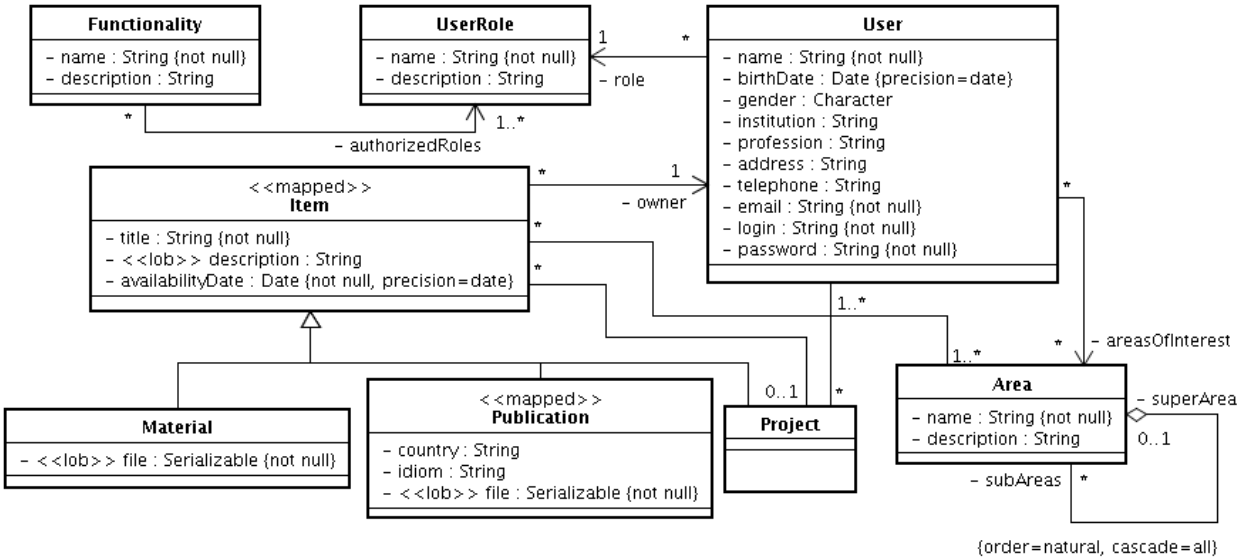

Figure 3. Domain model for the User Management module of LabES Portal.

configured to be implemented using natural ordering (a `SortedSet` is used) and to cascade all operations (eg. if an area is deleted, the ORM framework automatically deletes all subareas). `Item` and `Publication` are marked as mapped superclasses, which indicates that they are not persistent entities, but their attributes are persistent when inherited by other classes (`Publication`'s hierarchy isn't shown).

3.2.2. Persistence Model. The persistence model is an UML class diagram that shows the DAO [9] classes that compose the persistence layer. It guides the implementation of these classes and the creation of specific queries to the database (`Persistence` package). For each persistent class that needs persistence logic (i.e., needs to be stored, retrieved or deleted in an use case) the persistence model should have a DAO interface and one or more DAO classes (one for each persistence technology used). The DAO classes must implement the DAO interface, which in turn defines the methods available for the persistence of a certain class.

As a suggestion, all DAO interfaces should extend a `BaseDAO` interface, defining basic persistence methods that all DAOs should implement – retrieve all objects, retrieve an object given its id, save and delete an object. Given that these basic services will always be available, the DAO classes in the persistence model should display only methods for specific queries that are needed for the class they are responsible for persisting. For instance, to validate an user's login and password it will be necessary to retrieve all User objects with a given login attribute value. Therefore, `UserDAO` has the method `retrieveByLogin(login : String)` that indicates the existence of such query.

Besides the `BaseDAO` interface, the designer can also provide base DAO classes for any persistence technology used (eg. `HibernateBaseDAO` for Hibernate ORM framework). If provided, all concrete DAO classes of that technology (eg. `HibernateUserDAO`) can extend that base class.

Finally, since both interface and concrete classes are displayed in the diagram, there is no need to have both display their methods, since they are the same. The developer should choose one of them to display the methods and the other can remain empty. These rules reduce the amount of work, increasing productivity.

3.2.3. Navigation Model. The navigation model is an UML class diagram that depicts the different components that form the presentation tier for a given use case or scenario and their relationship among themselves. These components can be web pages (`<<page>>` stereotype), HTML forms (`<<form>>`), templates (`<<template>>`), binary files such as PDF files and images (`<<binary>>`) or action classes (no stereotype). Other classes can be shown in the diagram and the reader differentiates them from the action classes by naming conventions. Action classes belong to the `Controller` package, while the other Web resources are under `View`. Therefore, this model guides the construction of the presentation layer.

Figure 4 shows the navigation model for the Login scenario of the Authenticate User use case (from our example). On the `index` web page the user will find a form with login and password input fields, represented by `frmLogin` attributes. The types of the attributes indicate the types of HTML form fields to use and depend on the MVC framework used. When submitted, the information of the form is sent to the Front Controller, which will use an instance of `AuthenticateUserAction` to respond. That is depicted by the dependency association between the form and the action class.

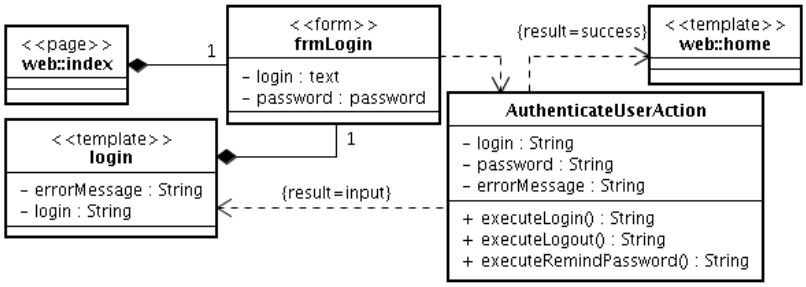

Figure 4. Navigation model for the Login scenario of Authenticate User use case (Portal LabES).

When receiving input from a form, the action class' attributes with the same name as the form fields receive their value before the action is executed. Executing the action consists of calling one of its execute methods. An action class can respond to many different use case scenarios and the method to be called is defined by the scenario's name (in this case, `executeLogin()`) or by a constraint in the incoming dependency association (eg. `{action=login}`).

The execution method returns a string indicating the result of the action. Dependency association from the action to templates or web pages indicate possible outcomes of the execution. If multiple outcomes exist, the associations must indicate which result they represent. In the example, if the login is successful, the user is sent to a web page generated by the template engine after processing the template `home`. If there are any problems with the input (unknown login, incorrect password, etc), the user is sent to another page to try logging in again.

The `login` template has a couple of attributes with the same name as the action class. That means that their values will be used in the template, namely to display what kind of problem occurred during authentication and to automatically fill the login form field with the last login used.

The navigation model can also represent links between web pages (using dependency associations) and the type of result to process after an action is executed, such as redirection, template processing and so on (represented by a constraint in the outgoing dependency association between the action class and some other element).

When building the navigation model, the developer must choose the granularity of the action class: one for each action, one for each use case scenario (a scenario can encompass more than one action), one for each use case (an use case can encompass more than one scenario) or one for various use cases. This model guides the construction of presentation tier components and the configuration of the MVC framework.

3.2.4. Application Model. The application model is an UML class diagram that displays classes from the `Application` package and their relationships with classes from the `Controller` and `Persistence` packages. It's not necessary to display the dependencies from `Application` classes with `Domain` classes because the associations with the DAO classes already demonstrate which classes are retrieved, saved or deleted from the database and, thus, are manipulated by the application class. Therefore, besides guiding the construction of the packages from the business tier, this model demonstrates how to integrate the different tiers to provide the desired solution in the proposed architecture.

Analogous to the granularity of the action classes in the navigation model, there is also a question of granularity for the application classes: we can have a different class for each scenario in an use case, one class for an entire use case, including all scenarios, or one class for many use cases. That being decided, the developer can systematically transform use cases or use case scenarios into methods of application classes, which will be made available to classes from the presentation layer that need to invoke those services.

For each application class modeled, there should be an interface and a concrete class, enforcing the "programming to interfaces" [16, 8] practice, which is highly recommended by the Dependency Injection frameworks. Since they define the same methods, there is no need to display them on both.

The application model guides the construction of application classes and also the configuration of the dependency injection framework, which will be responsible for wiring together all of these classes.

4. Related Work

The amount of propositions in the Web Engineering area, including methods, frameworks and modeling languages, is quite vast, demonstrating that academics and practitioners haven't yet elected a standard when it comes to Web development.

Conallen's work [4] is well known and defines a software process as well as a modeling language, named Web Application Extensions (WAE), that extends UML to provide Web-specific constructs for modeling WebApps. It advocates the construction of a new model, the User Experience (UX) Model, that defines guidelines for modeling layout and navigation from requirements specification through design. 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.
Both FrameWeb's modeling language and WAE define an UML profile for the creation of diagrams that picture web-related elements. However, as FrameWeb is based on frameworks, its stereotypes and constraints are different from those proposed by WAE. Also, we felt that a dependency association represents the relations among web components better than a regular association, which explains why FrameWeb is not an extension of WAE, but a new modeling language altogether.
As for Conallen's method, we find that FrameWeb introduces fewer new concepts, facilitating the adoption by developers already proficient in UML's most common diagrams. Also, Conallen's UX model spans all the way from requirements through design and FrameWeb proposes web-related models for the design activity only, allowing organizations to use their current processes without much change.
OOWS (Object Oriented Web Solution) [5] is a method for WebApp specification and development, which divides the process in two stages. In the first stage, structural, dynamic, functional, navigation and presentation models are built using UML. In the second stage, a component-based code generation strategy is used to automatically create an operational prototype using a conceptual model compiler.
OOWS 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. Its code generation strategy is something that FrameWeb still lacks and does provide a lot of agility to the process. Also, OOWS' navigation models define 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) [6] defines a set of models to be built, a modeling language that extends the UML meta-model, and a process to build the models using that modeling language. The process is composed by requirement analysis, conceptual navigation and presentation design, supplemented with task and deployment modeling. As with WAE, UWE's modeling language defines stereotypes, tagged values and constraints to create an UML profile specifically for Web components.
Although UWE relies on an UML profile with standard extensions, it does offer a few non-standard extensions that are available through the use of ArgoUWE, an extension of the ArgoUML[10] tool that supports UWE. As with WAE, it also doesn't define specific extensions for framework-related components. Similar to OOWS, UWE also supports a semi-automatic code generation solution that, as we noted before, is something that FrameWeb lacks. UWE is otherwise very similar to FrameWeb up to design stage, being based on use case and class diagrams.
Some works propose not a complete method but only a modeling language for the construction of Web-related models. WebML [3] proposes such a language, allowing developers to model WebApp's functionalities in a high level of abstraction, without committing to any architecture in particular. WebML is based on XML, but uses intuitive graphical representations that can easily be supported by a CASE tool. Its XML form is ideal for automatic generation of source code, producing Web applications automatically from the models.
Although its graphical representations are intuitive, not being based on UML is a big disadvantage of WebML for reasons of developer's acceptance and tool support. It does, however, provide the advantage of not being tied to any software process at all, letting organizations free to choose whatever method suits them.
A whole different category of propositions are hypermedia methods, such as OOHDM [7]. This kind of method focuses on contents and navigation structures instead of functionality and seems to be better suitable for information-driven WebApps.
Given all of the options available, FrameWeb comes in as another option that targets a specific architecture, one based in the use of frameworks. In this case, FrameWeb excels for its agility, because models are directed towards the framework architectures and allow for quick understanding of the implementation. It also doesn't introduce much complexity, allowing organizations to use their own processes up to design with few adaptations, if any. Of all the proposed design models, the navigation model is the only one we consider a little bit complex, making FrameWeb very easy to learn and use.
After CGI scripts and dynamic page technologies such as ASP, PHP and JSP, the use of framework-based architectures are becoming the standard for implementation of medium-to-large-sized WebApps. Taking the Java platform as example, the definition of standards as JavaServer Faces (JSF)[11] for Web development and the new Enterprise JavaBeans (version 3.0)[12] for distributed components reinforce that conclusion. JSF defines a MVC-like architecture, and EJB 3.0 had all of its persistence model reconstructed based on Hibernate ORM framework and also makes heavy use of Dependency Injection.
This context has motivated us for the creation of a modeling language for this specific case and finally led to the definition of a design method for WebApps.

5. Conclusions and future work

FrameWeb was applied in the development of the LabES Portal, as discussed along the paper. First, developers were trained in general concepts of Web Engineering, in

10 http://argouml.tigris.org/

11 http://jcp.org/en/jsr/detail?id=127
12 http://jcp.org/en/jsr/detail?id=220

the use of FrameWeb and also in the following frameworks: WebWork, FreeMarker (template engine), SiteMesh, Hibernate and Spring.

In general, the development went smoothly. The method allowed the developers to deliver the models mostly in time and few deadlines had to be extended.

Some developers had difficulties on capturing the idea of some frameworks, especially the MVC framework. All of them had some experience with the Java platform, but most did not have any experience with Web development.

At the end of the development, the developers were asked to provide feedback on the work done. This feedback can be summarized in the following items:

- Allowing to directly model aspects related to the use of frameworks is the biggest strength of FrameWeb;
- Implementing in Java what was modeled during design was very much facilitated by the clear understanding of the semantics of the four models (domain, persistence, navigation and application);
- The simplicity of the models facilitated the adoption of FrameWeb, except for the navigation model, which added some complexity to the method.

The method, while functional, could use a lot of improvement, some of which we list below:

- More case studies could be conducted to assess the effectiveness of the method. Some of FrameWeb's initial propositions have changed after the LabES Portal case study was done and many other improvements can come from more practical experiences;
- Other frameworks and implementation platforms could be studied in order to verify if the modeling language fits for the diverse options that exist. Our goal is to make the modeling language as generic as possible;
- Tools could be developed to help create the models or to convert the models to code, automatically implementing much of the infrastructure code and configuration for the most used frameworks available.

6. Acknowledgments

This work was accomplished with the financial support of FAPES, a foundation supporting science and technology from the government of the state of Espírito Santo – Brazil, and CAPES, an entity of the Brazilian Government reverted to scientific and technological development.

7. 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] Conte, T., Travassos, G. H., Mendes E., "Revisão Sistemática sobre Processos de Desenvolvimento para Aplicações Web". *Technical Report ESE/PESC* (in portuguese) – COPPE/UFRJ, Rio de Janeiro, 2005.

[3] Ceri, S., Fraternali, P., Bongio, A., "Web Modeling Language (WebML): a modeling language for designing Web sites". *Computer Networks*, Elsevier, June 2000, v. 33, n. 1-6, p. 137-157.

[4] Conallen, J., *Building Web Applications with UML*, 2nd edition, Addison-Wesley, October 2002.

[5] 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 (SCI)*, Orlando, FL – USA, July 2003.

[6] 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*, October 2000.

[7] Schwabe, D., Rossi, G., "An Object Oriented Approach to Web-Based Application Design". *Theory and Practice of Object Systems 4 (4)*, Wiley and Sons, 1998.

[8] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, October 1994.

[9] Sun Microsystems, "Core J2EE Patterns": http://java.sun.com/blueprints/corej2eepatterns/. Captured on July 19th, 2006.

[10] Fowler, M., "Inversion of Control Containers and the Dependency Injection Pattern": http://www.martinfowler.com/articles/injection.html. Captured on July 19th, 2006.

[11] Bauer, C., King, G., *Hibernate in Action*, 1st edition, Manning, August 2004.

[12] Souza, V. E. S., Falbo, R. A., "An Agile Approach for Web Systems Engineering". *Proceedings of the 11th Brazilian Symposium on Multimedia and the Web (WebMedia)*, ACM, Poços de Caldas, MG – Brazil, December 2005.

[13] Pressman, R.S., *Software Engineering: A Practitioner's Approach*, 6th edition, Mc Graw Hill, USA, April 2004.

[14] Ahmad, R., Li, Z., Azam, F., "Web Engineering: A New Emerging Discipline", *Proceedings of the IEEE Symposium on Emerging Technologies*, September 2005, 445-450.

[15] Ambler, S., Jeffries, R., *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, 1st edition, Wiley, March 2002.

[16] Schmidt, D., "Programming Principles in Java: Architectures and Interfaces", chapter 9: http://www.cis.ksu.edu/~schmidt/CIS200/