Edited By: Terry Halpin, John Krogstie, and Erik Proper, ISBN: 978-1-60566-278-7, IGI, 2009

Designing Web Information Systems for a Framework-based Construction

Vitor E. Silva Souza, Ricardo A. Falbo, Giancarlo Guizzardi

1. Introduction

The World Wide Web (also referred to as WWW or simply Web) was created as a means to publish documents and make them available to people in many different geographical locations. However, the advent of the Common Gateway Interface (CGI), in 1993, allowed for authors to publish software instead of documents and for visitors to execute them, producing dynamic results.

The evolution of Web development technology and the emergence of high-level languages (such as PHP, ASP, JSP, etc.) and platforms (such as Microsoft .NET and Java Enterprise Edition) allowed for more complex applications to be built on the Web. Soon enough, a handful of large B2C (business-to-consumer, such as online stores) and B2B (business-to-business, such as supply chain management systems) applications were being deployed on the Internet.

Thus, the concept of Web Applications (WebApps) was born. WebApps consist of a set of Web pages or components that interact with the visitor, providing, storing and processing information. WebApps can be informational, interactive, transactional, workflow-based, collaborative work environments, online communities, marketplaces or web portals (Ginige & Murugesan, 2001).

In this chapter, however, we focus on a specific class of Web Applications, called Web-based Information Systems (WISs). WISs are just like traditional information systems, although deployed over the Internet or on an Intranet. These systems are usually data-centric and more focused on functionality rather than content and presentation. Examples are online stores, cooperative environments, and enterprise management systems, among many others.

Although many Software Engineering principles have long been established before the creation of the Web, first-generation WebApps were constructed in an ad-hoc manner, with little or no concern for them. However, with the increase of complexity of the WebApps, which is especially true for WISs, the adoption of methodologies and software processes to support the development team becomes crucial.

Thus, a new discipline and research field was born. Web Engineering (or 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" (Murugesan et al., 1999, p. 2). Pressman (2005) complements this definition stating that WebE borrows many conventional Software Engineering fundamental concepts and principles and, in addition, incorporates specialized process models, software engineering methods adapted to the characteristics of this kind of application and a set of enabling technologies.

In this field, a lot of methods and modeling languages have been proposed. Some well known works are WebML (Ceri et al., 2000), WAE (Conallen, 2002), OOWS (Fons et al., 2003), UWE (Koch et al., 2000), and OOHDM (Schwabe & Rossi, 1998), among others.

Parallel to the academic research, the industry and the developer community have also proposed new technologies to provide a solid Web infrastructure for applications to be built upon, such as frameworks and container-based architectures. Using them we can improve productivity at the coding phase by reusing software that has already been coded, tested and documented by third parties. As their use becomes state-of-the-practice, methods that focus on them during software design could provide a smoother transition from models to source code.

This has motivated us to develop a WebE design method that focuses on frameworks. The Framework-based Design Method for Web Engineering (FrameWeb) (Souza & Falbo, 2007) proposes a basic architecture for developing WebApps and a UML profile for a set of design models that brings concepts used by some categories of frameworks, which are applied in container-based architectures as well.

Meanwhile, many researches have been directed to the construction of what is being considered the future of the WWW: the Semantic Web. Coined by Berners-Lee et al. (2001), the term represents an evolution of the current WWW, referred by some as the "Syntactic Web". In the latter, information is presented in a way that is accessible only to human beings, whereas in the former data is presented both in human-readable and machine-processable formats, in order to promote

the development of software agents that would help users carry out their tasks on the Web.

However, for Berners-Lee's vision to become a reality, Web authors and developers must add semantic annotations to their Web Applications. This is neither an easy nor a small task and support from tools and methods is needed. Thus, an extension of FrameWeb was proposed. The Semantic FrameWeb (S-FrameWeb) (Souza et al., 2007) incorporates into the method activities and guidelines that drive the developer in the definition of the semantics of the WISs, resulting in a "Semantic Web-enabled" application.

The objective of this chapter is to discuss the current research state regarding Web Engineering (methods and modeling languages), frameworks and the Semantic Web, and to present FrameWeb, and its extension S-FrameWeb, as a new method based on best practices for the development of Web-based Information Systems. We close the chapter by presenting future trends and opportunities for this research.

2. Background

Web Engineering was born from the need to apply Software Engineering principles to the construction of WebApps, adapting them to the application's size, complexity and non-functional requirements. A lot of research on methods and modeling languages has already been conducted, providing an extensive background for new research.

Meanwhile, companies and independent developers create frameworks and propose container-based architectures to promote reuse and improve productivity while maintaining good design principles. Furthermore, research on the Semantic Web has been pointing out some directions on what the Web may become in the future.

This section discusses the current state-of-the-art and state-of-the-practice on Web Engineering, modeling languages for WebE, frameworks for development of WebApps and the Semantic Web.

2.1. Web Engineering

Web Engineering (or WebE) uses scientific, engineering, and management principles and systematic approaches to successfully develop, deploy, and maintain high-quality WebApps (Murugesan et al., 1999).

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 coded using tools specialized for the Web. Finally, the system is tested and delivered to end-users (Pressman, 2005).

Considering that, in general, the platform in which the system will run is not taken into account before the design phase of the software 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 (Ahmad et al., 2005), such as sensitivity to content, short time frames for delivery, continuous evolution, focus on aesthetics, etc (Pressman, 2005).

This has motivated researchers to propose different methods, modeling languages and frameworks for Web Engineering. The amount of propositions is quite vast, demonstrating that academics and practitioners have not yet elected a standard concerning Web development. In this subsection we briefly present some methods, while the following subsections focus on modeling languages and frameworks.

Web Application Extension (WAE) (Conallen, 2002) defines an iterative and incremental software process, centered on use cases and based on the Rational Unified Process (Krutchen, 2000) and the ICONIX Unified Process (Rosenberg & Scott, 1999). It proposes activities such as business analysis, project planning, configuration management and an iterative process that includes the usual software development cycle from requirement gathering to deployment.

OOWS (Object Oriented Web Solution) (Fons et al., 2003) is an extension of the OO-Method (Pastor et al., 2001) for WebApp specification and development. It divides the software process in two main steps: conceptual modeling and solution development. In the conceptual modeling step, the system specification is obtained by using conceptual models. For that, OOWS introduces new models for representing navigational and presentational characteristics of web applications. In the

solution development step, the target platform is determined, and a specific architectural style is chosen. Then, a set of correspondences between abstraction primitives and the elements that implement each tier of the architectural style are applied in order to automatically obtain the final system (Pastor et al., 2003).

The UML-based Web Engineering (UWE) (Koch et al., 2000) is a development process for Web applications with focus on systematic design, personalization and semi-automatic generation. It is an object-oriented, iterative and incremental approach based on the Unified Modeling Language (UML) and the Unified Software Development Process (Jacobson et al., 1999). The notation used for design is a "lightweight" UML profile. The process is composed by requirement analysis, conceptual navigation and presentation design, supplemented with task and deployment modeling and visualization of Web scenarios (Koch & Kraus, 2002).

Lee & Shirani (2004) propose a component-based methodology for WebApp development, which is divided in two major parts: component requirements analysis and component specifications. Analysis begins identifying the required component functions and is followed by a comparison with the functions available in existing components. The component specification phase has three activities: rendering specification, integration specification and interface specification.

The Ariadne Development Method (Díaz et al., 2004) proposes a systematic, flexible, integrative and platform-independent process for specification and evaluation of WebApps and hypermedia systems. This process is composed of three phases: conceptual design, detailed design and evaluation. Each phase is further subdivided into activities, which in turn defines sets of work products to be built.

Díaz et al. (2004, p. 650) also define the hypermedia paradigm as one that "relies on the idea of organizing information as a net of interrelated nodes that can be freely browsed by users selecting links and making use of other advanced navigation tools, such as indexes or maps". We consider hypermedia methods quite different than methods for the development of WISs, as they focus on content and navigational structures instead of functionality and seem to be better suitable for information-driven WebApps.

Although hypermedia development methods are not on our focus, it is worthwhile to cite OOHDM (Object Oriented Hypermedia Design Method) (Schwabe & Rossi, 1998), a well-known method that is representative of hypermedia methods. It was born from the need to represent hypermedia structures such as links, text-based interfaces and navigation, and more recently has also been applied to Web development. For instance, an extension of this method, called OOHDM-Java2 (Jacyntho et al., 2002), was proposed, which consists of a component-based architecture and an implementation framework for the construction of complex WebApps based on modular architectures (e.g. Java EE). The OOHDM process is divided into five steps: requirements gathering, conceptual design, navigational design, abstract interface design and implementation.

During our research we have also found several other methodological approaches that target specific contexts or scenarios, such as:

- The Business Process-Based Methodology (BPBM) (Arch-int & Batanov, 2003), which blends advantages of the structured and object-oriented paradigms for identifying and designing business components. The central idea of business component modeling is reusability of elementary units, which are business activities. An elementary unit that represents an atomic changeable business process can be implemented with a portable set of Web-based software components;
- The Internet Commerce Development Methodology (ICDM) (Standing, 2002), which is focused on the development of B2C e-commerce applications, emphasizing not only technical aspects, but also strategic, business and managerial aspects.

Some of the methods presented above also propose a modeling language that better suits its purposes, such as WAE and UWE . In the next subsection, some of them are briefly presented.

2.2. Modeling Languages for Web Engineering

Modeling languages define notations to be used on the creation of abstract models to solve problems. The Unified Modeling Language (UML) (Booch et al., 2005), for instance, is a modeling language that defines on its metamodel standardized notations for different kinds of models, such as class diagrams and use case diagrams. However, UML does not define when and to which

purpose each model should be used.

Hence, methodologies usually present their own modeling language or, as is most commonly seen, use and extend UML, defining a UML Profile. For this purpose, UML includes extension mechanisms, such as stereotypes (definition of a new model element based on an existing one), tagged values (attachment of arbitrary textual information to elements using label/value pairs) and constraints (semantic specification for an existing element, sometimes using a formal language).

Based on these extension mechanisms, Conallen (2002) proposed the Web Application Extensions (WAE), which extends UML to provide Web-specific constructs for modeling WebApps. WAE also advocates the construction of a new model, the User Experience (UX) Model, which defines guidelines for layout and navigation modeling 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.

The UML-based Web Engineering (UWE) (Koch et al., 2000) also defines a UML profile. Based on class and association elements, it defines new elements to describe Web concepts, such as navigation, indexes, guided tours, queries, menus and many others.

Another method that defines a modeling language based on UML is OOWS (Fons et al., 2003). For the construction of the navigational model, UML packages represent navigational contexts and form a directed graph where the arcs denote pre-defined valid navigational paths. Each context is further modeled using a class diagram to show the navigational classes that form them.

Not all modeling languages are UML-based. WebML (Ceri et al., 2000) is an example. It allows 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.

Methods and modeling languages aid developers mostly during analysis and design of information systems. However, one can also find tools that focus on the implementation phase. In the next subsection, we discuss frameworks that have been extensively used for the development of WISs.

2.3. Frameworks for Web Development

WISs have very similar architectural infrastructure. Consequently, after the first systems started to be built, several frameworks that generalize this infrastructure were developed to be reused in future projects. In this context, a framework is viewed as a code artifact that provides ready-to-use components that can be reused via configuration, composition or inheritance. When combined, these frameworks allow large-scale n-tier WISs to be constructed with less coding effort.

Putting together several of these frameworks can produce what we call a container-based architecture. A container is a system that manages objects that have a well-defined life cycle. A container for distributed applications, such as the applications servers for the Java Enterprise Edition (Shannon, 2003), manage objects and offer services such as persistence, transaction management, remoting, directory services, etc.

The use of these frameworks or container-based architectures has a considerable impact in the development of a WIS. Since it is possible to find many frameworks for the exact same task, we categorized them according their objectives into the following classes:

- Front Controller (or MVC) frameworks;
- Decorator frameworks;
- Object/Relational Mapping frameworks;
- Dependency Injection frameworks;
- Aspect-Oriented Programming frameworks;
- Authentication & Authorization frameworks.

2.3.1. Front Controller Frameworks

MVC stands for Model-View-Controller (Gamma et al., 1994). It is a software architecture that was developed by the Xerox PARC for the Smalltalk language in 1979 (Reenskaug, 1979) and has found great acceptance by Web developers. When applied to the Web, the MVC architecture is adapted and receives the name "Front Controller" (Alur et al., 2003, p.166). Both terms are used indistinguishably by Web developers.

The Front Controller architecture is depicted in Figure 1. When structured in this architecture, a WebApp manages all requests from clients using an object known as Front Controller. Based on a customizable configuration, this object decides which class will respond to the current request (the action class). Then, following the Command design pattern (Gamma et al., 1994), it instantiates an object of that class and delegates the control to it, expecting some kind of response after its execution. Based on that response, the controller decides the appropriate view to present as result, such as a web page, a report, a file download, among other possibilities.

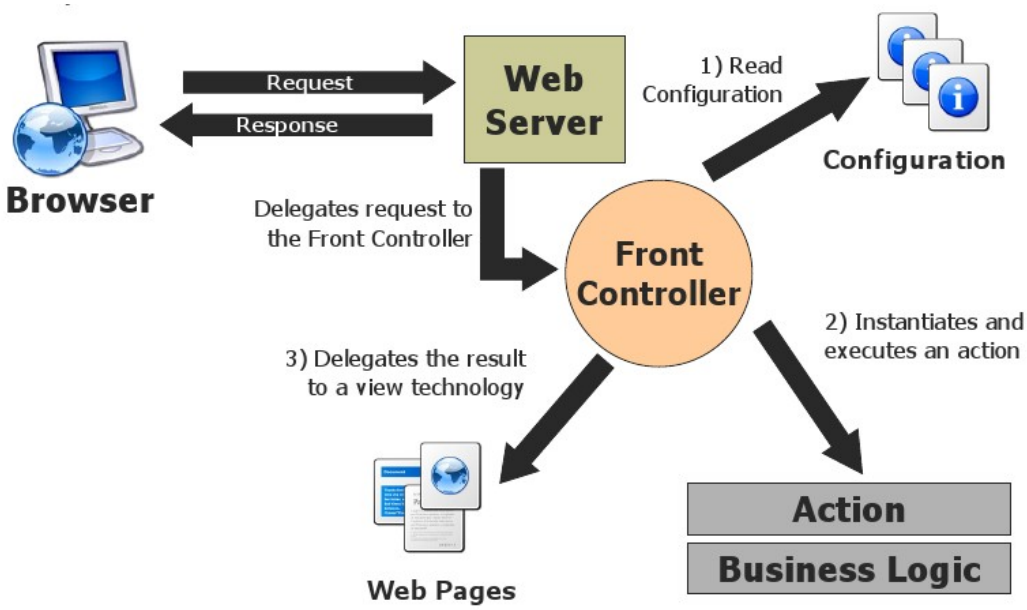

Figure 1: general architecture of a Front Contoller framework.

One of these possibilities is using a template engine that defines a template language that is usually more suitable for the view layer than the usual dynamic web technology (such as JSP, ASP or PHP). The template language is usually simpler, making it possible for Web Designers without specific programming skills to build them. Also, they tend to help developers not to break the MVC architecture by restricting what can be done in the template language (e.g. can not directly connect to a database from a template).

MVC Frameworks usually provide the front controller, a super-class or interface for action classes, several result types and a well defined syntax for the configuration file. The template engine is a separate tool, but the framework usually provides integration to it. Note that on n-tier applications, this framework belongs to the Web tier and should delegate business and persistence tasks to components on appropriate tiers.

Only for the Java platform, for instance, there are more than 50 MVC frameworks. Some of the most popular are Struts[1], Spring MVC[2] and Tapestry[3].

2.3.2. Decorator Frameworks

Decorator frameworks automate the otherwise tedious task of making every web page of the site have the same layout, meaning: header, footer, navigational bar, color schemes and other graphical layout elements produced by a Web design team. Figure 2 shows how a decorator framework works.

1 http://struts.apache.org/2.x/index.html
2 http://www.springframework.org
3 http://tapestry.apache.org

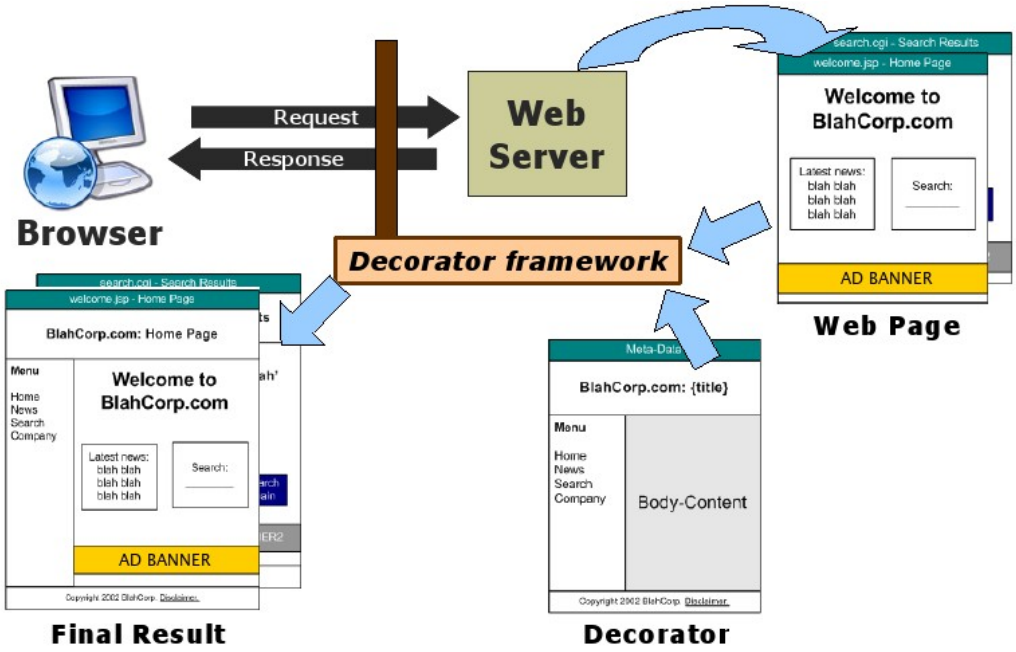

Figure 2: the process of decoration of websites.

They work like the Decorator design pattern (Gamma et al., 1994), 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" of the page. Examples of this kind of framework are SiteMesh[4] and Tiles[5].

2.3.3. Object/Relational Mapping Frameworks

Relational Database Management Systems (RDBMS) have long been the *de facto* standard for data storage. Because of its theoretical foundations (relational algebra) and strong industry, even object oriented applications use it for object persistence, giving rise to a "paradigm mismatch" (Bauer & King, 2004): tables, rows, projection and other relational concepts are quite different from a graph of interconnected objects and the messages they exchange.

Among the many options to deal with this problem, there is the Object/Relational Mapping (ORM) approach, shown in Figure 3, which is the automatic and transparent persistence of objects to tables of a RDBMS using meta-data that describe the mapping between both worlds (Bauer & King, 2004). Instead of assembling a string with the SQL command, the developer provides mapping meta-data for the classes and call simpler commands, such as `save()`, `delete()` or `retrieveById()`. An object-oriented query language can also be used for more complex retrievals.

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 Java ORM framework is Hibernate[6]. Other well-known frameworks are Java Data Objects[7], Apache Object Relational Bridge[8] and Oracle Toplink[9].

[4] http://www.opensymphony.com/sitemesh
[5] http://struts.apache.org/struts-tiles
[6] http://www.hibernate.org
[7] http://java.sun.com/products/jdo
[8] http://db.apache.org/ojb/
[9] http://www.oracle.com/technology/products/ias/toplink

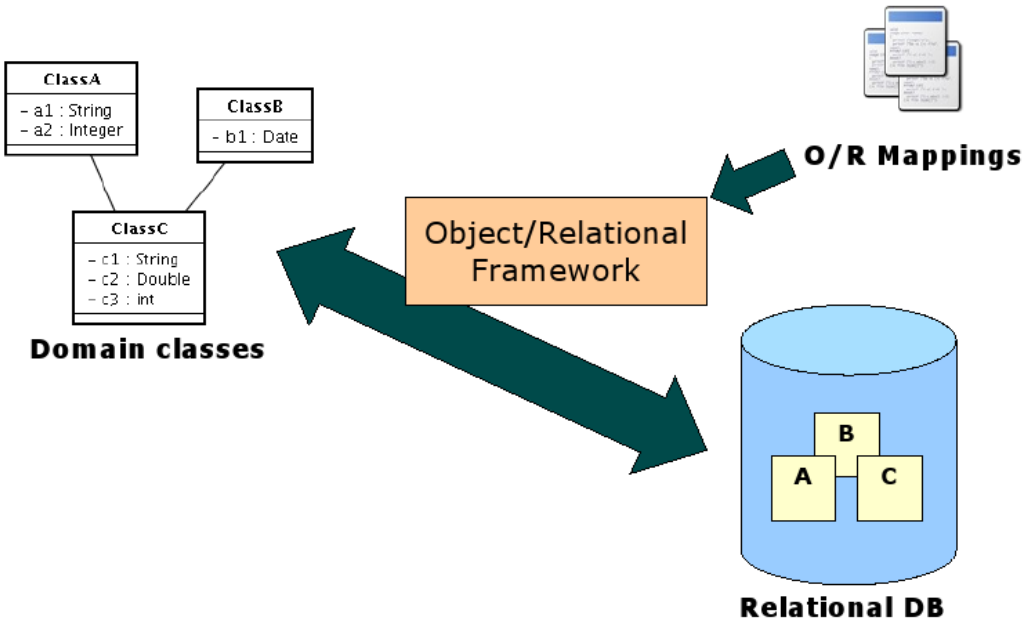

Figure 3: persistence of objects using an ORM framework.

2.3.4. Dependency Injection Frameworks

Object-oriented applications are usually built in tiers, each of which having a separate responsibility. According to Fowler (2007), 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.

Creational design patterns, such as Factory Method, Abstract Factory and Builder (Gamma et al., 1994), help implementing this good practice in programming, known today as "programming to interfaces, not implementations" (Schmidt, 2007). For instance, if a service class depends on a data access class, it does not need to know *how* the data access class will perform its duty, but only *what* it will do and what method should be called for the job to be done.

Dependency Injection (DI) frameworks allows the developer to program to interfaces and specify the concrete dependencies as meta-data in a configuration file. When a certain object is obtained from the DI framework, all of its dependencies are automatically injected and satisfied. An abstract example is shown in Figure 4: when the client asks for an instance of `SomeClass`, the DI framework first satisfies `SomeClass`' dependencies and delivers the object with all dependencies fullfiled – in the example, an instance of `DependencyClass`.

These frameworks are also known as Inversion of Control (IoC) frameworks, since the control (who creates the objects) is removed from the dependent classes and given to the framework. As well as ORM frameworks, DI frameworks are not used 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, PicoContainer[10], Apache Hivemind[11], etc.

[10] http://www.picocontainer.org
[11] http://jakarta.apache.org/hivemind

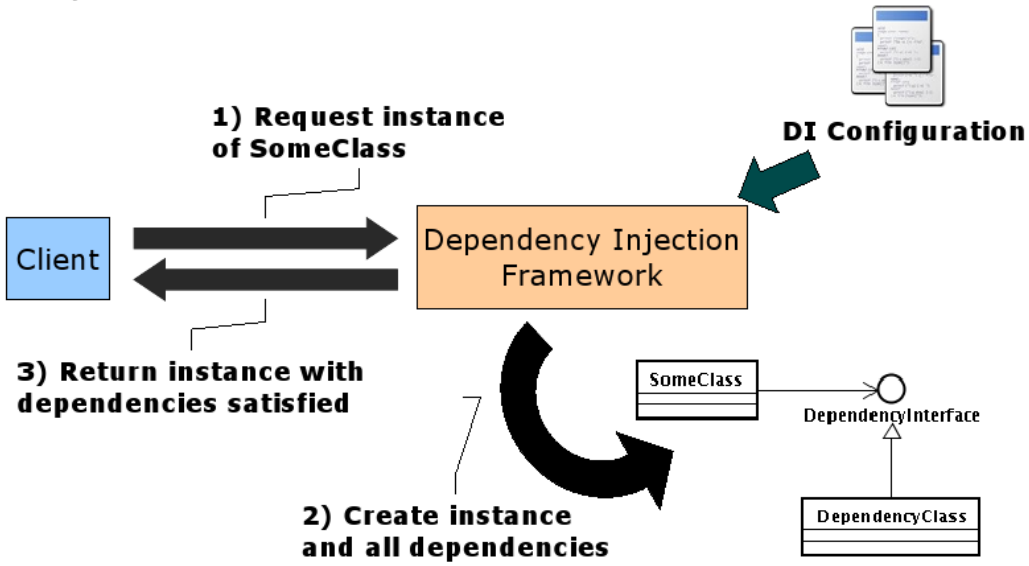

Figure 4: dependency injection using a framework.

2.3.5. Aspect-Oriented Programming Frameworks

The Aspect-Oriented paradigm is based on the concept of separation of concerns: the idea is to separate different concerns of a system to be treated separately, thus reducing the complexity of development, evolution and integration of software (Resende & Silva, 2005). Although it concerns the whole development process, its biggest influence is at the coding phase, with Aspect Oriented Programming (AOP).

Once a *cross-cutting concern* is identified (e.g.: logging, transaction management), instead of repeating similar code in different points, the functionality can be implemented in a single place, becoming an *aspect*. Then, the different places where that aspect should be applied are identified (these are called *pointcuts*) and, before the code is executed, a process called *weaving* is conducted to automatically spread the aspect all over the code.

The weaving can be conducted by an AOP framework during runtime or by an AOP compiler during compilation time. Many infrastructure concerns that are usual in Web applications are good candidates for this separation, making AOP frameworks very popular. One example, depicted in Figure 5, is that of transaction management. An AOP framework can make all business methods transactional with few configuration steps, avoiding the effort of repeatedly implementing the same logic in all of them.

Some well-known AOP frameworks for the Java platform are AspectJ[12], Spring Framework and JBoss AOP[13].

[12] http://www.eclipse.org/aspectj
[13] http://labs.jboss.com/portal/jbossaop

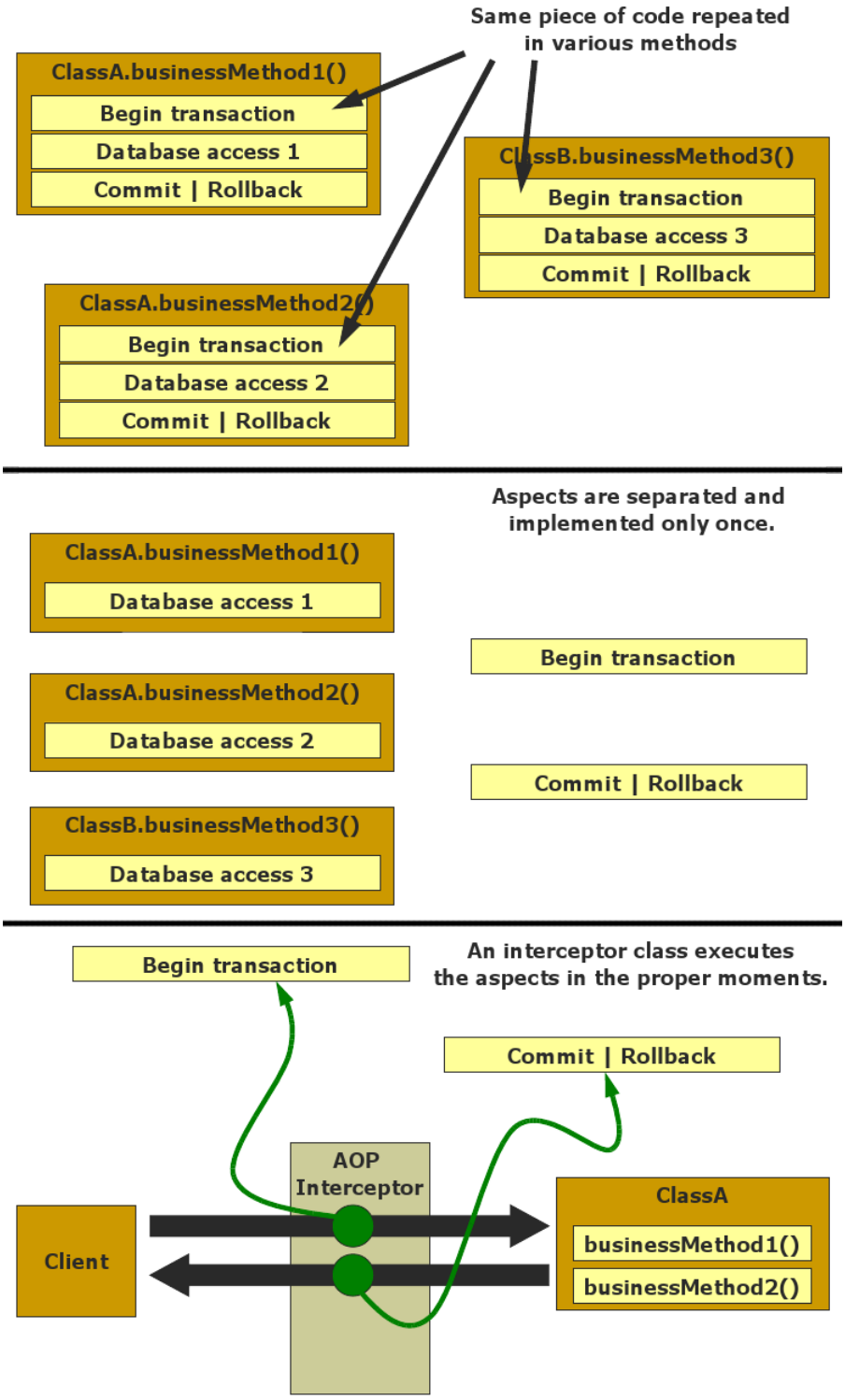

Figure 5: example of application of AOP using an AOP runtime framework.

2.3.6. Authentication & Authorization Frameworks

Another common concern of Web information systems is that of guaranteeing the security of the information. This is usually done by two different procedures: authentication (verifying if an access key is valid to access the application) and authorization (verifying the level of access of the authenticated user and what she is allowed to do).

Being such an important task, frameworks were created to guarantee its proper execution. They can be configured to support many different "auth" methods, using, as usual, meta-data and configuration files. Some well-known auth frameworks for the Java platform are Acegi Security for

Spring[14], Apache Cocoon Authentication[15] and the Java Authentication and Authorization Services[16].

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 proposed FrameWeb, a Framework-based Design Method for Web Engineering (Souza & Falbo, 2007), which is presented in section 3.

2.4. The Semantic Web

The Semantic Web is being proposed as an evolution of the current WWW, in which information is provided both in human-readable and computer-processable formats, in order to allow for the semi-automation of many tasks that are conducted on the Web.

In order for the software agents to reason with the information on the Web (reasoning meaning that the agents are able to understand it and take sensible actions according to a predefined goal), web pages have to be presented also in a machine-readable form. The most usual way for this is annotating the pages using formal knowledge representation structures, such as ontologies.

An ontology is an engineering artifact used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of its vocabulary words (Guarino, 1998). Along with ontology representation languages such as OWL (W3C, 2007a), they are able to describe information from a website in formal structures with well-defined inference procedures that allow software agents to perform tasks such as consistency checking, to establish relationships between terms and to systematically classify and infer information from explicitly defined data in this structure.

Designing an ontology is not a straightforward task. There are many methodologies for their construction (Gomez-Perez et al., 2005) and attention has to be given to the selection of concepts, their properties, relationships and constraints. However, after the ontology is built, the annotation of static Web pages with languages such as OWL becomes a simple task, especially with the aid of tools, such as OILEd[17] and Protégé[18].

However, several websites have their Web pages dynamically generated by software retrieving information from data repositories (such as relational databases) during runtime. Since these pages cannot be manually annotated prior to their presentation to the visitor, another approach has to be taken. Two approaches that have been proposed are dynamic annotation and semantic Web services.

The former works by recognizing whether the request belongs to a human or a software agent, generating the proper response depending on the client: in the first case, a HTML human-readable Web page; in the second, a document written in an ontology specification language containing meta-data about the information that would be displayed in the HTML version. Although the solution seems appropriate, many aspects still need to be addressed, such as: how are the agents supposed to find the Web page? How will they know the correct way to interact with it? For instance, how will they know how to fill in an input form to submit to a specific request?

The latter approach is based on Web Services, which are software systems designed to support interoperable machine-to-machine interaction over a network (W3C, 2007b). Web Services provide a nice way for software agents to interact with other systems, requesting services and processing their results. If semantic information is added to the services, they could become interpretable by software agents. Meta-data about the service are written in a markup language, describing its properties and capacities, the interface for its execution, its requirements and the consequences of its use (McIlraith et al., 2001). Many tasks are expected to be automated with this, including service discovery, invocation, interoperation, selection, composition and monitoring (Narayanan & McIlraith, 2002).

As the research on the Semantic Web progresses, methods are proposed to guide developers on building "Semantic Web-enabled" applications. An example of this is the Semantic Hypermedia Design Method (SHDM) (Lima & Schwabe, 2003). Based on OOHDM (Schwabe & Rossi, 1998), SHDM is a comprehensive model-driven approach for the design of Semantic WebApps.

---

[14] http://www.acegisecurity.org
[15] http://cocoon.apache.org
[16] http://java.sun.com/products/jaas
[17] http://oiled.man.ac.uk/
[18] http://protege.stanford.edu/

SHDM's process is divided in 5 activities. In the first step, Requirements Gathering, requirements are gathered in the form of scenarios, user interaction diagrams and design patterns. The next phase, Conceptual Design, produces a UML-based conceptual model, which is enriched with navigational constructs in the Navigational Design phase. The last two activities are Abstract Interface Design and Implementation.

2.5. Some Considerations

In our research, we haven't found a method focused on the use of frameworks for the construction of WISs nor for the development of Semantic Web applications. In the next sections, we present FrameWeb, our proposal for the design of framework-based WISs, and its extension, S-FrameWeb, which incorporate into the method activities and guidelines that drive the developer in the definition of the semantics of the WISs, resulting in a "Semantic Web-enabled" application.

3. FrameWeb

FrameWeb is a design method for the construction of Web-based Information Systems (WISs) based on frameworks. The main motivations for the creation of this method were:

(a) The use of frameworks or similar container-based architectures has become the *de facto* standard for the development of distributed applications, especially those based on the Web;

(b) There are many propositions in the area of Web Engineering, including methodologies, design methods, modeling languages, frameworks, etc. However, we haven't found one that deals directly with the particularities that are characteristic of the use of frameworks;

(c) Using a method that fits directly into the architecture chosen for the implementation promotes a greater agility to the software process, which is something that is desired in most Web projects.

In general, FrameWeb assumes that certain types of frameworks will be used during the implementation, defines a basic architecture for WISs and proposes design models that are closer to the implementation of the system using these frameworks.

Being a design method, it doesn't prescribe a complete software process. However, it suggests the use of a development process that includes the following activities, as presented in Figure 6: requirements elicitation, analysis, design, coding, testing and deployment. For a more systematic usage of the method, it also suggests that, during Requirement Elicitation and Analysis, use case diagrams are used to model requirements and class diagrams are used to represent the conceptual model.

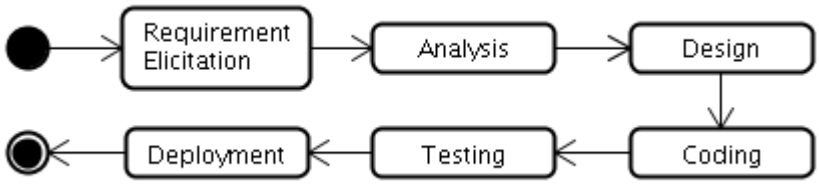

Figure 6: A simple software process suggested by FrameWeb.

Also, as mentioned earlier, one of the motivations for the creation of FrameWeb is the demand for agility that surrounds Web 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 (Ambler & Jeffries, 2002).

The main contributions of the method are for the Design phase: (i) the definition of a basic architecture that divides the system in layers with the purpose of integrating better with the frameworks; (ii) a UML profile for the construction of four different design models that bring the concepts used by the frameworks to the design stage of the software process.

The Coding phase is facilitated by the use of frameworks, especially because design models show

components that can be directly related to them. The use of frameworks can also have impacts on Testing and Deployment, but these are yet subject to study and research.

Throughout the next subsections we detail FrameWeb's basic architecture and its UML profile. Examples diagrams were taken from the development of a portal for the Software Engineering Lab (LabES) of the Federal University of Espírito Santo State using FrameWeb. Figure 7 shows its use case diagram, simplified for brevity.

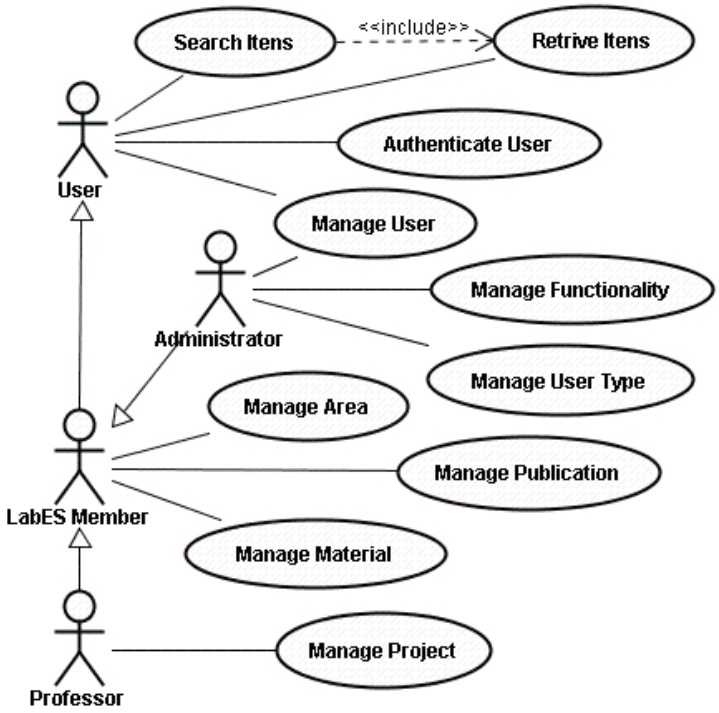

Figure 7: A simplified use case diagram for LabES Portal.

The "LabES Portal" was proposed to provide a better interaction with the Software Engineering community. This WIS has a basic set of services providing information about current LabES projects, areas of interest, publications and other material available for download. Figures 8 and 9 show the conceptual models produced during Analysis.

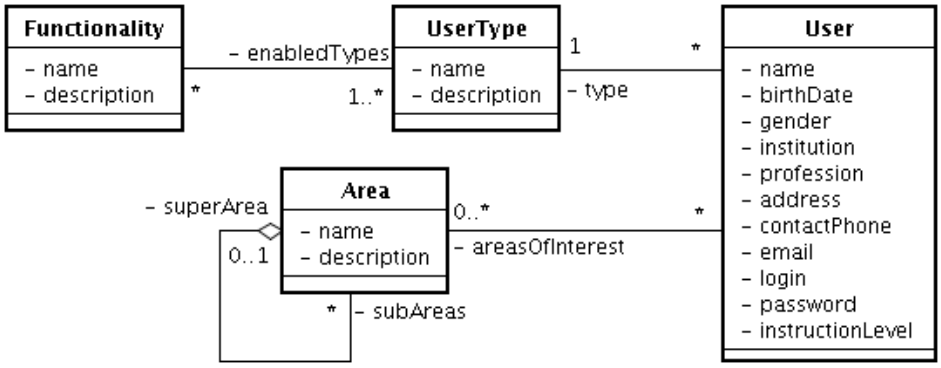

Figure 8: Conceptual model for the User Control module of the LabES Portal.

Basically, the portal makes a collection of items available. These items can be organized in projects and subprojects or belong to the lab in general. Publications (papers, books, book chapters and academic work) and generic materials can be published in the portal. Items are also related to users (responsible user, editing users) and areas of interest.

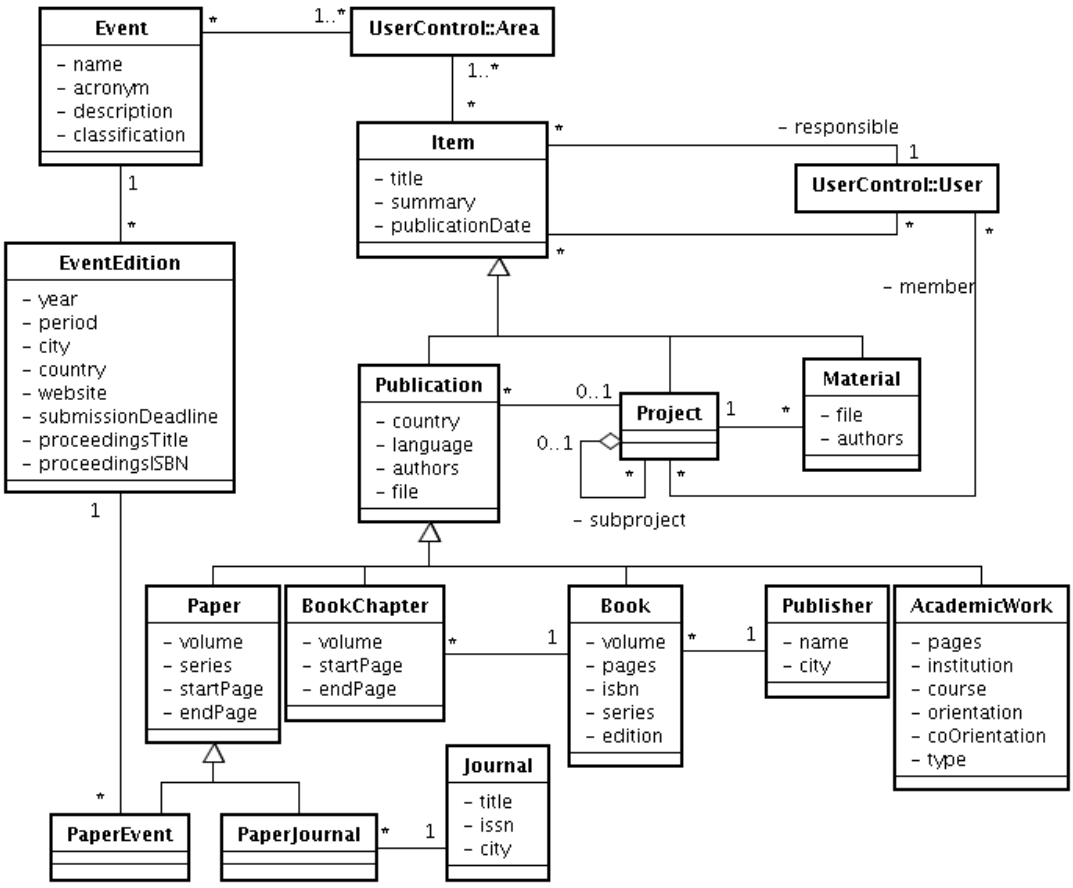

Figure 9: Conceptual model of the Item Control module of the LabES Portal.

3.1. Framework-based WebApp Architecture

The Design activity, traditionally executed after requirement elicitation and analysis, has as purpose the description of the logical and physical architectures of the system as well as the development of structural and behavioral models built based on the models developed in the previous phases, but that now consider the specific characteristics of the chosen implementation platform.

FrameWeb defines a logical architecture for WISs based on the architectural pattern Service Layer (Fowler, 2002, p. 133). As depicted in Figure 10, the system is divided in three main layers: presentation logic, business logic and data access logic.

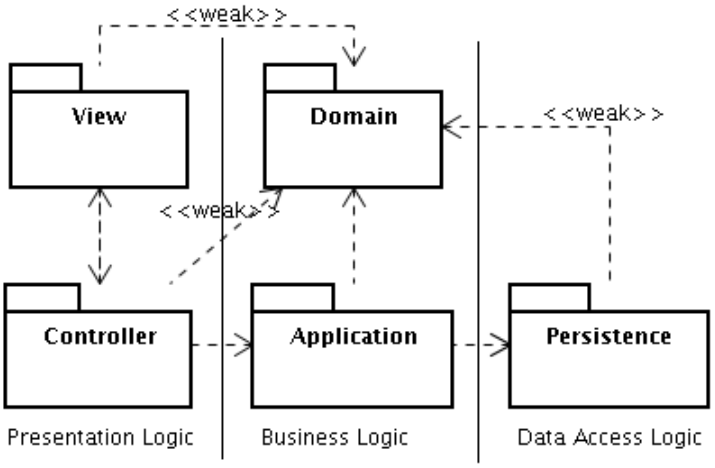

Figure 10: FrameWeb's basic architecture for WISs.

The first layer concerns the graphical user interfaces. The `View` package contains the Web pages, style sheets, images, client-side scripts, templates and everything else related to the exhibition of information to the user. The `Controller` package encompasses action classes and other files related to the Front Controller framework. These two packages are mutually dependent, since `View` elements send user *stimuli* to `Controller` classes while these process the response using pages, models and other `View` components.

The business logic is implemented in the second layer, divided in two packages: `Domain` and `Application`. The former contains classes that represent concepts of the problem domain identified and modeled by the class diagrams during analysis and refined during design. The latter has the responsibility of implementing the use cases defined in the requirements specification, providing a service layer independent of the user interface. The `Application` classes deal directly with `Domain` objects to implement system functionality and, thus, this dependency is represented in the diagram.

The `Controller` package, on the presentation layer, depends on the `Application` package since it mediates the user access to the system functionalities. User *stimuli* coming from `View` are transformed by the `Controller`'s classes in method calls to classes in the `Application` package. `Controller` and `View` have also dependency relationships with `Domain`, but this is tagged as weak to denote low coupling: `Domain` objects are used only for exhibition of data or as parameters on method invocations between one package and another, i.e., the presentation layer does not have the right to alter domain entities.

The third and last layer regards data access and has only the `Persistence` package. This package is responsible for the storage and retrieval of persistent objects in long-term duration media, such as databases, file systems, naming services, etc. In the case of FrameWeb, it expects the use of an ORM framework through the Data Access Object (DAO) pattern (ALUR et al., 2003, p. 462). The DAO pattern adds an extra abstraction layer, separating the data access logic of the chosen persistence technology in a way that the `Application` classes do not know which ORM framework is being used, allowing for its replacement, if necessary. It also facilitates unit testing, as one can provide mock DAOs for the `Application` classes to be tested alone.

As we can see in Figure 10, the `Application` package depends on the `Persistence` package to retrieve, store and delete domain objects as the result of use case execution. Since the `Persistence` package works with `Domain` objects, a weak dependency is also portrayed in the figure.

This architecture provides a solid base for the construction of WISs based on the types of frameworks presented in subsection 2.3. Each package contains classes or other elements that integrate with these frameworks and, to model all these elements, FrameWeb proposes a modeling language based on the UML, which is presented next.

3.2. Modeling Language

During design, besides specifying the system architecture, the artifacts that will be implemented by the programmers on the coding phase should be modeled. Since FrameWeb is based on the frameworks presented in subsection 2.3, we felt the need for a modeling language that would represent the concepts that are present in these frameworks.

Following the same approach as other modeling languages such as WAE and UWE, FrameWeb uses UML's lightweight extensions to represent typical Web and framework components, creating a UML profile that is used for the construction of four kinds of diagrams, which are presented in the following subsections: domain model, persistence model, navigation model and application model.

3.2.1. Domain Model

The domain model is a UML class diagram that represents domain objects and their persistence mapping to a relational database. This model is used by the programmers to implement the classes of the `Domain` package. FrameWeb suggests its construction in two steps:

1. Adapt the conceptual model produced during the Requirement Analysis phase to FrameWeb's architecture and to the chosen platform of implementation. This requires choosing data types for attributes, defining navigabilities of the associations, promoting attributes to classes (if necessary), etc.;

2. Add persistence mappings.

Persistence mappings are meta-data that allow ORM frameworks (see subsection 2.3.3) to convert objects in memory to tuples in Relational Data Base Management Systems and vice-versa. Mappings are added to the domain model using stereotypes and constraints that guide developers in the configuration of the ORM framework during implementation. Despite the fact that these mappings are more related to persistence than domain, they are shown in this model because the classes that are mapped and their attributes are shown here.

Table 1 describes the possible O/R mappings for the domain model. For each mapping, the table presents the extension mechanism used and what are its possible values or syntax. None of the mappings is mandatory and most of them have sensible defaults, reducing the amount of elements that have to be modeled. The default values are shown in the third column in boldface.

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=*name* **(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)
If the attribute represents the versioning column.	Attribute stereotype	<<version>>
If an attribute should be stored in a large object field (e.g.: CLOB, BLOB)	Attribute stereotype	<<lob>>
Name of the column in which an attribute will be persisted	Attribute constraint	column=*name* **(defaults to the attribute's name)**
Size of the column in which an attribute will be persisted	Attribute constraint	size=*value*
If the association should be embedded (instead of having its own table, the associated child class' attributes are placed in the parent's table)	Attribute stereotype	<<embedded>>
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 | *column names* [asc | desc])
Cascading of operations through the association: nothing, persists, merges, deletions, refreshs or all	Association constraint	cascade = (**none** | persist | merge | remove | refresh | all)
Association fetching strategy: lazy or eager.	Association constraint	fetch = (lazy | **eager**)

Table 1: Possible OR mappings for the Domain Model.

The Domain Model for the User Control module of sLabES Portal is shown in Figure 11. According

to the default values, all classes are persistent and class and attribute names are used as table and column names respectively.

As we can see in the diagram, attributes have received mappings such as nullability and size. The `birthDate` attribute was mapped as date-only precision. The recursive association in Area was configured to be sorted naturally (will be implemented in the programming language) and to cascade all operations (e.g. if an area is deleted, all of its subareas are automatically deleted).

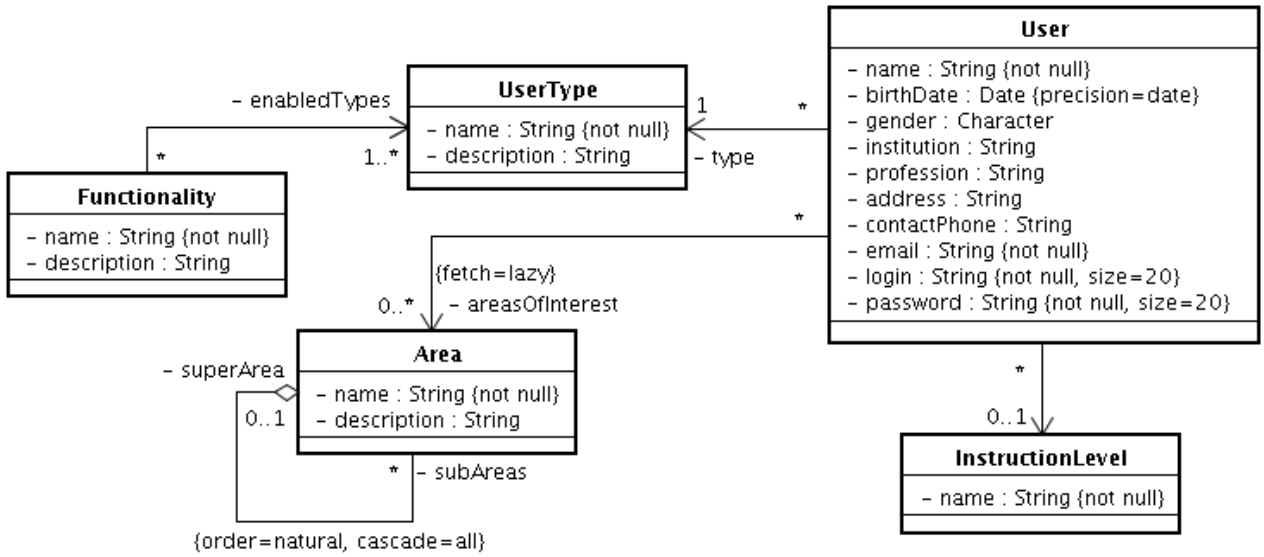

Figure 11: Domain Model for the User Control module of LabES Portal.

None of the classes have ID or version attributes because they are inherited from a utility package, as shown in Figure 12. The `mapped` stereotype indicates that `DomainObjectSupport` and `HibernatePersistentObject` are not persistent entities, but their subclasses, which are entities, inherit not only their attributes but also their O/R mappings. All domain classes in sLabES Portal are said to extend `HibernatePersistentObject`, inheriting, thus, the UUID[19], the persistence ID and the version attribute.

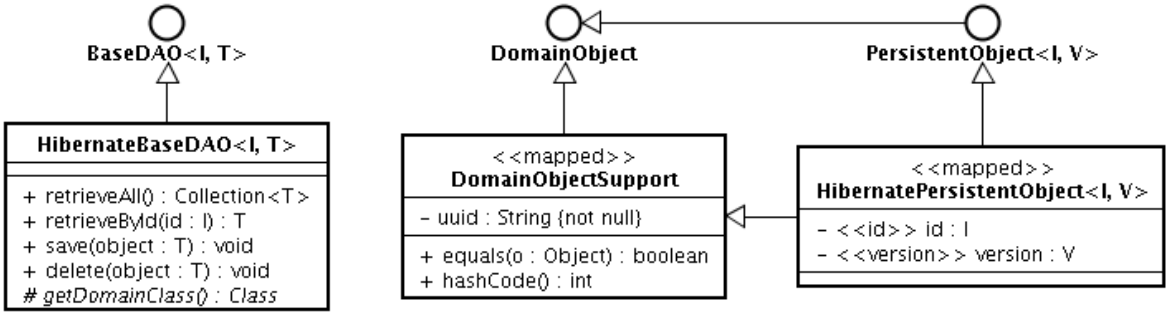

Figure 12: Utility classes for persistence in JSchool.

The parameters `I` and `V` are generic, allowing for the user to choose the type of ID and version attributes. `HibernateBaseDAO` is a base class for data access objects, described in the persistence model, discussed in the next subsection.

3.2.2. Persistence Model

As mentioned before, FrameWeb indicates the use of the DAO design pattern (ALUR et al., 2003, p.

[19] The relationship between an object's identity in memory and its primary key in the database raises several issues that are discussed in the article "Hibernate, null unsaved value and hashcode: A story of pain and suffering" from Jason Carreira (http://www.jroller.com/page/jcarreira?entry=hibernate_null_unsaved_value_and). The idea of using a Universal Unique Identifier (UUID) was taken from this article.

462) to the construction of the data access layer. Thus, the persistence model is a UML class diagram that represents DAO classes responsible for the persistence of the domain classes. Therefore, it guides the implementation of the classes from the `Persistence` package. FrameWeb suggests three steps for its construction:

1. Model the interface and concrete implementation of the base DAO (an example from the JSchool project is shown in Figure 12);
2. Define which domain classes need basic persistence logic and create a DAO interface and implementation for each one;
3. For each DAO, evaluate the need of specific database queries, adding them as operations in their respective DAOs.

The persistence model presents, for each domain class that needs data access logic, an interface and a concrete DAO that implements the interface. The interface has to be unique and defines the persistence methods for a specific domain class. One concrete class is modeled for each persistence technology used.

To avoid repeating in each DAO operations that are common in all of them (e.g.: save, delete, retrieve by ID, etc.), a Base DAO (interface and implementation class) is modeled in a utility package. Automatically all DAO interfaces inherit from the BaseDAO interface and the same happens with concrete implementations, without the need to explicitly state that in the diagram. Also, to avoid repeating methods in the interface and implementations, the designer can choose to display them in one of the two only and it is inferred that all public methods are defined in the interface and implemented in the concrete class.

Figure 12 shows the interface and implementation using Hibernate ORM framework, designed for the sLabES Portal project. Both interface and class and declared using generic types, leaving to their subclasses to specify which class is being persisted and what is the type for its ID attribute. The Base DAO defines methods to retrieve all persistent entities of a given class, retrieve an entity given its ID, save and delete an entity. As stated before, all public methods modeled in `HibernateBaseDAO` are inferred to be defined in the `BaseDAO` interface.

Figure 13 shows the modeling of four DAOs from the sLabES Portal project, for the persistence of the classes in the User Control module. `AreaDAO` and `UserTypeDAO` are simple, as they inherit all basic operations from the Base DAO and don't need to define any extra ones. The other two define extra operations. For example, UserDAO defines an operation to retrieve all users that have a given area of interest. This is necessary because there is no navigability from Area to User (see Figure 11) and the “Manage Area” use case needs to prevent an area from being deleted if it is associated with any user.

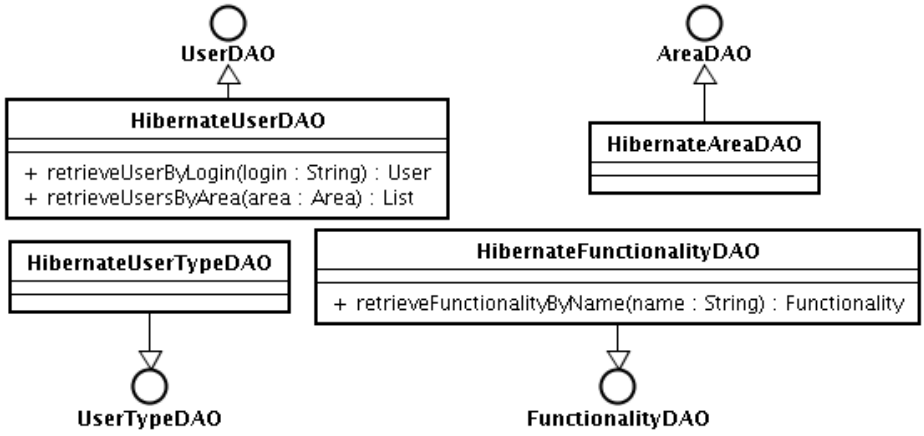

Figure 13: Persistence model of the User Control module of LabES Portal.

As we can see, the persistence model does not define any UML extensions to represent the concepts that are needed to implement the data access layer, but only some rules that make this modeling simpler and faster.

3.2.3. Navigation Model

The navigation model is a UML class diagram that represents different components that form the presentation layer, such as Web pages, HTML forms and action classes from the Front Controller framework (see subsection 2.3.1). Table 2 shows the UML stereotypes used by the different elements that can be represented in a navigation model. This model is used by developers to build classes and components of the `View` and `Controller` packages.

Stereotype	What it represents
(none)	An action class, to which the Front Controller framework delegates the execution of the action.
`<<page>>`	A static or dynamic Web page.
`<<template>>`	A template that is processed by a template engine and is transformed into a Web page.
`<<form>>`	A HTML form.
`<<binary>>`	Any binary file that can be retrieved and displayed by the browser (e.g.: images, reports, documents, etc.).

Table 2: UML stereotypes used in the navigation model.

For Web pages and templates, the attributes of the classes represent information from the domain that is supposed to be displayed in the page. Dependency relationships between them indicate hyperlinks while composition associations between pages and forms denote the presence of the form in that page.

In HTML forms, attributes represent the form fields and their types follow the HTML standard for types of fields (e.g.: input, checkbox, etc.) or the names of the JSP tags used by the framework (e.g., for Struts[2], textfield, checkbox, checkboxlist, etc.).

The action class is the main component of the model. Its dependency associations show the control flow when an action is executed. Table 3 lists the different meanings of this kind of association, depending on the components that are connected by it. Dependencies that are navigable towards an action class represent method calls, while the others represent results from the action execution.

From	To	What it represents
Page / template	Action class	A link in the page/template that triggers the execution of the action.
Form	Action class	Form data are sent to the action class when the form is submitted.
Action class	Page / template	The page/template is shown as one of the results of the action class.
Action class	Binary file	A binary file is shown as one of the results of the action class.
Action class	Action class	An action class is executed as result of another. This process is known as “action chaining”.

Table 3: Dependency associations between an action class and other elements

The attributes of the action class represent input and output parameters relevant to that action. If there is a homonymous attribute in an HTML form being submitted to the action, it means that the data is injected by the framework in the action class (input parameter). Likewise, when one of the result pages/templates show an attribute with the same name of an attribute of the action class,

this indicates that the framework makes this information available for the output.

When an action is executed, the framework will execute a default action method or allow/request the explicit definition of which method to execute. In the latter case, the designer must specify which method is being executed using the constraint `{method=method-name}` in the dependency association. The same is true for associations that represent results. Naturally, these methods should be modeled in the diagram.

When modeling action chaining, it's sometimes necessary to indicate the method that was executed in the first action and the one that will be executed in the following. These can be specified with the constraints `outMethod` and `inMethod`.

For dependency associations that represent results there are two other constraints that can be used:

- `{result=result name}` specifies a keyword that represents this control flow, i.e., when the action class returns this keyword as result of the action execution, the framework will follow this flow and show the appropriate result page/template/binary file;
- `{resultType=type name}` determines the type of result, among those supported by the framework. Usually, at least the following types of result are available: `binary` (display a binary file), `chain` (action chaining), `dispatch` (dispatches the request), `redirect` (redirects the request) and `template` (processes a template using a template engine).

The difference between a dispatch and a redirection is that the first makes the action's output parameters available to the view, while the second does not. When a dependency association doesn't specify a type, it means it is a `dispatch`. The default result is defined by the framework.

The designer is free to choose the granularity of the action classes, building one for each use case scenario, one for each use case (encompassing many scenarios), one for multiple use cases, and so forth. Moreover, he/she should decide if it's best to represent many actions in a single diagram or have a separate diagram for each action. Figure 14 is the navigation model for a use case of sLabES Portal.

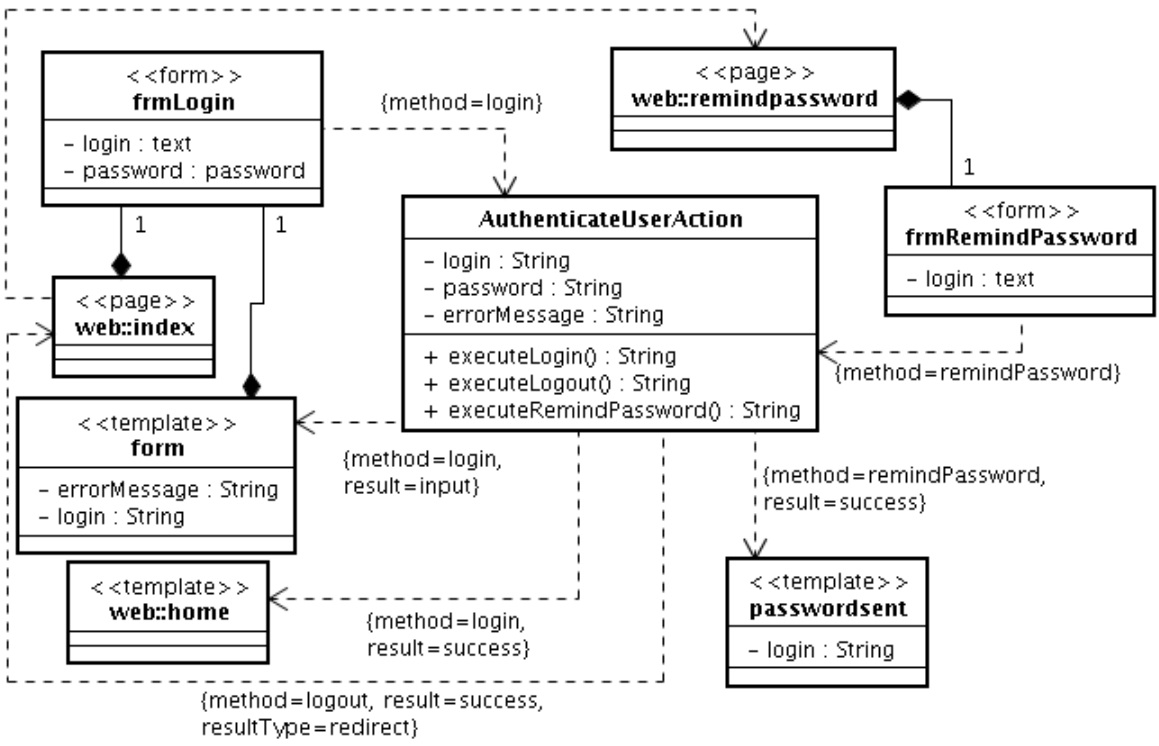

Figure 14: Navigation Model for the use case "Autheticate User"

The figure shows that in the initial page of the portal (represented by `web::index`), there is a form where login and password can be filled. When submitted, this data goes to the action class for the execution of the `executeLogin()` method, which would access the business logic layer to perform the use case. If the information filled is correct (`result = success`), the user is taken to web::home, which represents the starting page for authenticated users. Otherwise, the user will be taken back to `web::index` (`result = input`), showing once again the login form and an error message.

If the user forgot his/her password, he/she can click on a link in the initial page to go to the `web::remindpassword` page, where his/her login would be informed and sent to the action class. The `executeRemindPassword()` method requests the business logic layer to send the password to the user's email address and informs the user that the message has been sent. To log out, the user clicks on the appropriate link and is redirected back to the initial page.

During the conception of FrameWeb, there has been a discussion on whether the navigation model would be better represented by a sequence diagram, as it could represent better the control flow. Two main reasons led to the choice of the class diagram: (a) it provides a better visualization of the inner elements of action classes, pages and forms; and (b) it models composition between pages and forms with a more appropriate notation. Nonetheless, designers are advised to build sequence diagrams to represent complex flows when they see fit.

Last but not least, FrameWeb suggests four steps for the construction of a navigation model:

1. Study the use cases modeled during requirements analysis to define the granularity of the action classes (using, preferably, names that can relate the actions to the use cases/scenarios they refer to);
2. Identify how the data gets to the action class, modeling input pages and forms and the appropriate attributes on them and in the action class;
3. Identify what are the possible results and model the output pages/templates/binary files, also adding attributes when appropriate. We suggest that results that come from exceptions should not be modeled to avoid polluting the diagram;
4. Periodically check if the model is getting too complex and consider dividing it into two or more navigation models.

3.2.4. Application Model

The application model is a UML class diagram that represents classes from the `Application` package and their relationship with the `Controller` and `Persistence` packages. Besides guiding the implementation of application classes, this diagram also instructs developers on the configuration of the Dependency Injection framework (see section 2.3.4), which is responsible for managing the dependencies among these three packages.

The granularity of the application classes can be chosen by the developer in the same way as the granularity of the action classes. The application model also shares similarities with the persistence model, as it does not define any UML extension and uses the “programming to interfaces” principle, indicating the modeling of an interface for each application class.

When an application class is modeled, all action classes that depend on it should be displayed in the diagram, with the appropriate namespaces and relationships depicted. Analogously, all DAOs required by the application class to execute the use case should have their interfaces shown in the model, along with the relationship with the application class. Both relationships are represented by directed associations and the multiplicity is not required, as it is always 1.

Figure 15 shows part of an application model of sLabES Portal, depicting the classes that implements the “Manage User” and “Authenticate User” use cases and its relationships with controller and persistence components. The methods of the classes represent each scenario of each use case and define the parameters that should be given for them.

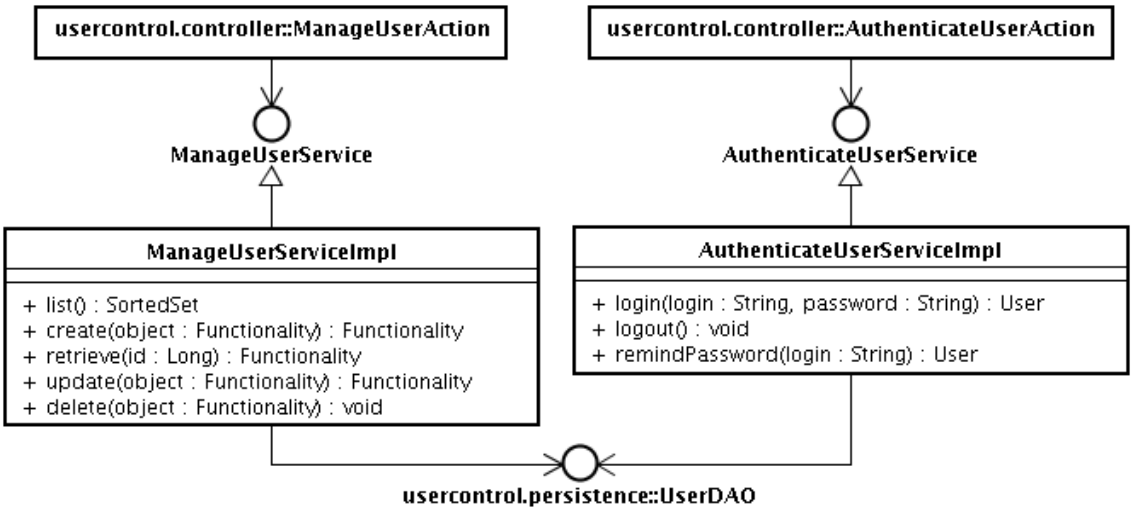

Figure 15: Part of an Application Model of the User Control module of LabES Portal.

Application classes manipulate domain objects and, thus, depend on them. These relationships, however, are not shown in the diagram to avoid increasing the complexity of the model. One can know about these relationships by reading the description of each use case.

FrameWeb suggests four steps for the construction of an application model:

1. Study the use cases modeled during analysis to define the granularity of the application classes (using, preferably, names that can relate the classes to the use cases/scenarios they implement);
2. Add to the interfaces/classes the methods that implement the business logic, giving special attention to the name of the method (as before, with the name of the class), its parameter, the parameters types and its return type;
3. By reading the use case descriptions, identify which DAOs are necessary for each application class and model the associations;
4. Go back to the navigation model (if already built) and identify which action classes depend on which application class and model their associations.

By defining the standard architecture and a UML profile for the construction of these four diagrams, FrameWeb provides the appropriate tools for the design of framework-based WISs. To promote the construction of "Semantic Web-enabled" WISs, an extension called S-FrameWeb was proposed and it is presented in the next section.

4. S-FrameWeb

The main goal of S-FrameWeb is to make WISs "Semantic Web-enabled". Being a framework-centered method, the chosen approach is to have the Front Controller framework produce dynamic annotations by identifying if requests come from human or software agents. In the former case, the usual Web page is presented, while in the latter, an OWL document is returned.

To accomplish this, S-FrameWeb extends FrameWeb in the following manners:

- The activity of Domain Analysis should be conducted in the beginning of the project to build an ontology for the domain in which the software is based. If it already exists, it should be reused (and eventually modified);
- Requirement Specification and Analysis go as usual, except for the fact that conceptual models build during Analysis can now be based on the domain ontology built in the

previous activity;

- During design, FrameWeb's Domain Model (FDM) receives semantic annotations based on the domain ontology;
- During implementation, the MVC framework has to be extended in order to perform dynamic annotation.

Figure 16 shows the software process suggested by S-FrameWeb while Table 4 summarizes the evolution of the models throughout that software process.

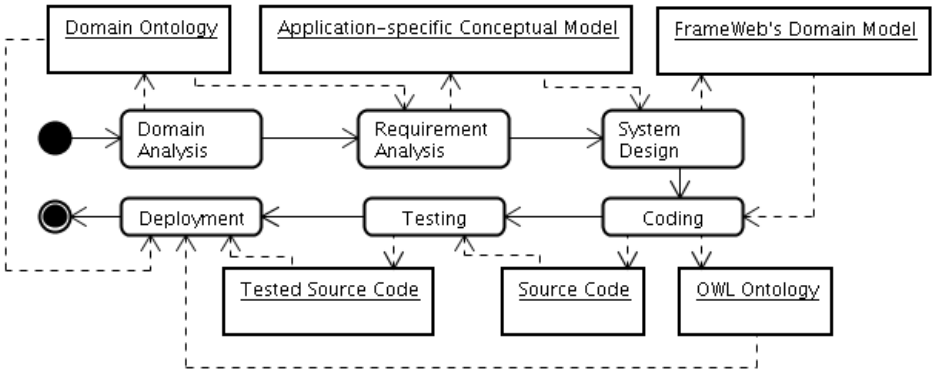

Figure 16: The software process suggested by S-FrameWeb (SOUZA et al., 2007).

Activity	Artifact	What the model represents
Domain Analysis	Domain Ontology	Concepts from the domain to which the software is being built. Modeled in ODM, but converted to OWL for deployment.
Requirement Analysis	Conceptual Model	Concepts that are specific to the problem being solved. Modeled in ODM.
System Design	FrameWeb's Domain Model (FDM)	Same as above plus OR mappings. Modeled using S-FrameWeb's UML profile.
Coding	OWL code	OWL representation of FDM, without OR mappings.

Table 4: Models produced by the software process suggested by S-FrameWeb (SOUZA et al., 2007)

The following subsections go through the suggested software process discussing it in more detail.

4.1. Domain Analysis

The first step for bringing a WIS to the Semantic Web is formally describing its domain. As discussed in section 2.4, this can be achieved by the construction of an ontology. S-FrameWeb indicates the inclusion of a Domain Analysis activity in the software process for the development of a domain ontology (we don't use the term “domain model” to avoid confusion with FrameWeb's Domain Model (FDM), which is a design model).

Domain Analysis is “the activity of identifying the objects and operations of a class of similar systems in a particular problem domain” (Neighbors, 1981; Falbo et al., 2002). When a software is built, the purpose is to solve a problem from a given domain of expertise, such as medicine, sales or car manufacturing. If the domain is analyzed prior to the analysis of the problem, the knowledge that is formalized about the domain can be reused when another problem from the same domain needs a software solution (Falbo et al., 2002).

S-FrameWeb does not impose any specific method for the construction of ontologies. It also doesn't require a specific representation language, but suggests the use of OMG's[20] Ontology

[20] Object Management Group – http://www.omg.org/ontology/

Definition Metamodel (ODM) (OMG, 2007), “a language for modeling Semantic Web ontologies in the context of MDA” (Đurić, 2004). ODM defines an ontology UML profile that allows developers to represent ontologies in UML class diagrams.

In the development of the LabES Portal, the SABiO method (Falbo, 2004) was followed, resulting in the construction of an ontology for educational portals that deals with competency questions such as: what are the roles of the people in the educational institution?, what are the areas of interest of these people and the institution?, how is the institution organized?, etc. The ontology was divided into two separate diagrams: one for the general structure of educational protals and another specific for publications. Figure 17 shows the first one.

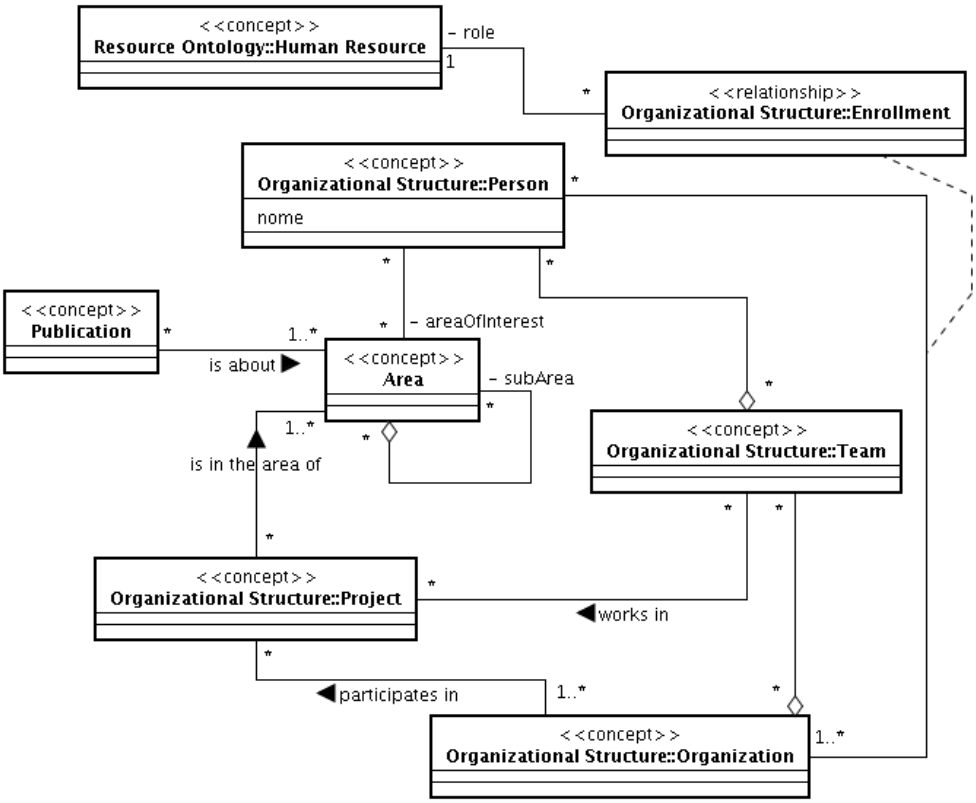

Figure 17: Diagram of the structural part of the ontology for educational portals.

The domain ontology serves as a basis for the construction of the application's conceptual model (during Requirement Analysis), which should derive some classes and associations from the ontology, adding and modifying elements as needed, concerning the specific problem being solved.

4.2. Requirement Specification and Analysis

The activities of Requirement Specification and Analysis should be conducted by the development team using its methodology of preference. S-FrameWeb, like FrameWeb, does not prescribe any methods or languages to this phase of the software process. However, as during Domain Analysis, it suggests the use of ODM for the graphical representation of the conceptual model, as it eases its conversion to FDM and, later on, to code (using OWL).

Figure 18 shows the conceptual model for the User Control module of the LabES Portal.

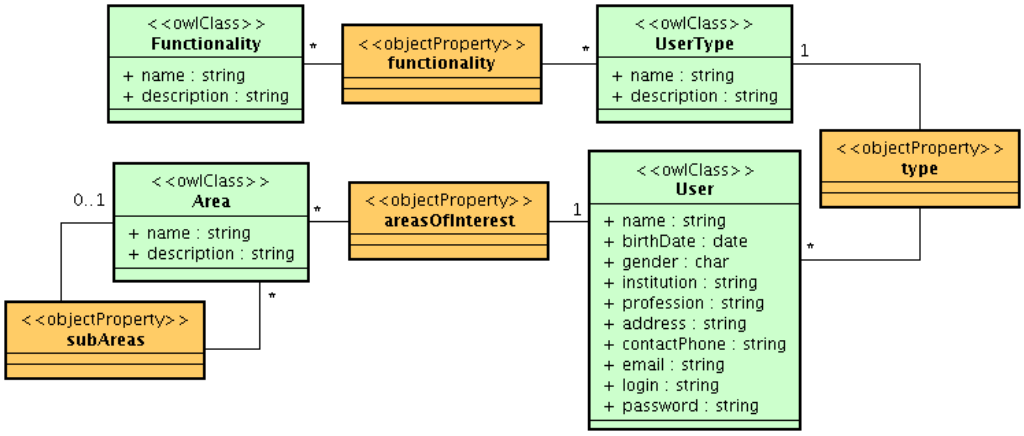

Figure 18: The conceptual model for the User Control module of LabES Portal, in ODM.

The stereotype `<<OntClass>>` indicates domain classes, `<<ObjectProperty>>` models associations between domain classes, `<<DataType>>` represents XML data types and `<<DatatypeProperty>>` models associations between classes and data types.

The reader accustomed with UML conceptual models may notice that associations are represented as classes in ODM. This is because in OWL associations are independent from classes and, for instance, can form their own subsumption hierarchy. This could also happen with attributes, for the same reasons. More on ODM's semantics can be found at (OMG, 2007).

In the cases where there is no need to represent associations or attributes as UML classes, S-FrameWeb suggests the conceptual model is simplified, such as the one shown in Figure 19. Notice that this diagram is very similar to the one in Figure 8.

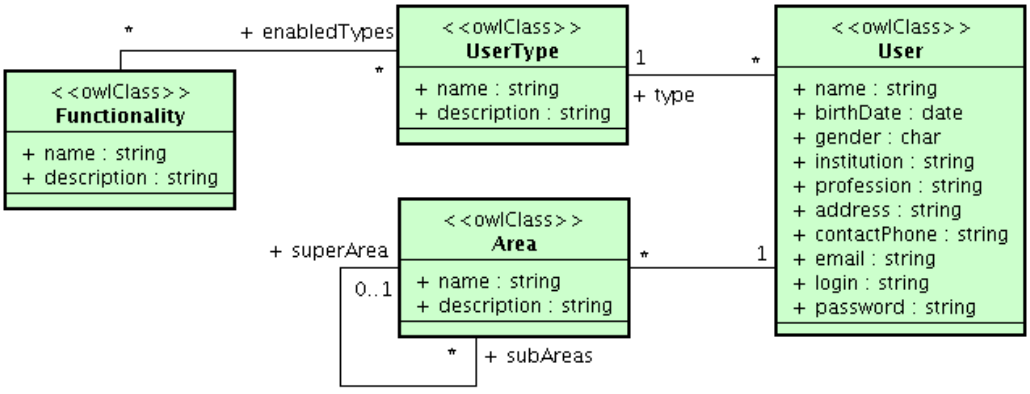

Figure 19: The conceptual model for the User Control module of LabES Portal, in its simplified version.

4.3. Design

As discussed in section 3.2, FrameWeb proposes the creation of four kinds of models during design: domain, persistence, navigation and application models. These models are still used with S-FrameWeb, although the domain model (FDM) should be adapted to a representation more suitable to the purposes of this semantic extension. Therefore, S-FrameWeb suggests a new UML profile for this diagram, mixing the profile defined by ODM with the one proposed by FrameWeb.

This new profile consists basically of the one defined by ODM, with the following adaptations:

1. Specification of association navigabilities for the implementation of the classes;
2. Addition of the O/R mappings for the configuration of the ORM framework;

3. Use of the data types of the implementation platform instead of those defined by the XML Schema Definition (XSD) standard[21];
4. Simplification of ODM's syntax when possible (if not already done previously).

Naturally, the construction of the FDM should be based on the conceptual model already built in previous activities. Figure 20 shows the FDM for the LabES Portal. We can see that, based on the simplified version of the conceptual model, association navigabilities were defined, data types were chosen among those of the implementation platform and that some O/R mappings were included. The result is very similar of that of Figure 11, due to the simplifications performed.

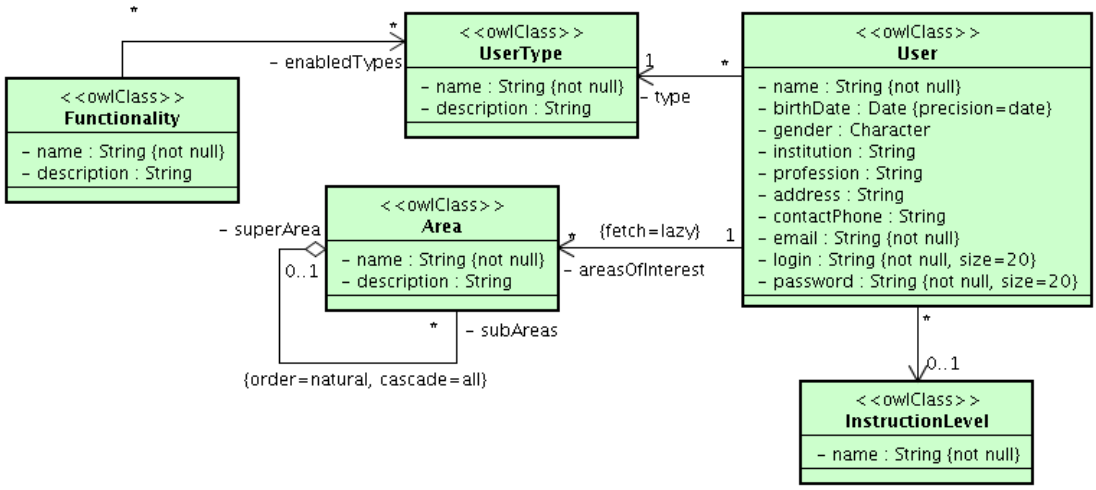

Figure 20: S-FrameWeb's Domain Model for the User Control module of LabES Portal.

The representation of this model in a language that mixes profiles from both ODM and FrameWeb attempts to facilitate the implementation phase, when an OWL file representing the conceptual model should be created and the ORM framework should be configured.

4.4. Implementation, Testing and Deployment

During implementation, the classes that, integrated with the frameworks, provide a software solution to the problem at hand are developed. S-FrameWeb adds a new task to this activity: the construction of OWL files representing the domain ontology and the application conceptual model (based on the FDM). As stated before, this task is facilitated by the use of ODM in both models.

The OWL files should be used by the Front Controller framework to implement dynamic annotation on the Web pages. S-FrameWeb proposes an extension to this kind of framework that recognizes when a request comes from a human or from a software agent by analyzing a specific HTTP request parameter (e.g. `owl=true`). In the case of a software agent, the framework should respond with an OWL file that is based on the domain ontology and the conceptual model, and represents the data that would be shown in the human-readable version of the page.

To experiment this approach in practice, a prototype of an extension for the Struts[2] framework was built. Figure 21 shows this extension and how it integrates with the framework. The client's web browser issues a request for an action to the framework. Before the action gets executed, the controller automatically dispatches the request through a stack of interceptors, following the pipes and filters architectural style.

[21] The XML Schema standard can be found at http://www.w3.org/XML/Schema. Its data types are described in a specific page, at http://www.w3.org/TR/xmlschema-2.

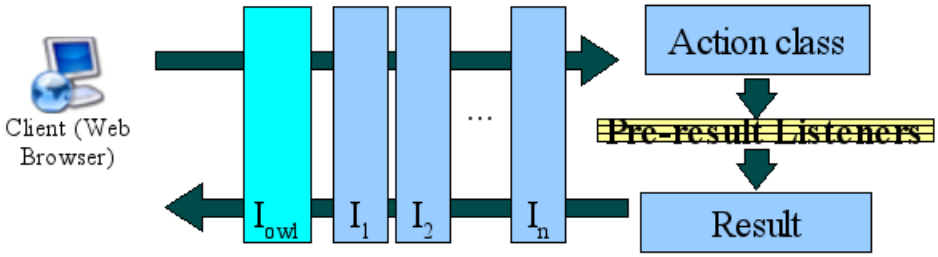

Figure 21: S-FrameWeb's Front Controller framework extension for the Semantic Web (SOUZA et al., 2007).

An "OWL Interceptor" was developed and placed as first of the stack. When the request is made, this interceptor verifies the HTTP parameter and, if present, creates a pre-result listener that will deviate successful requests to the "OWL Result Class", another custom-made component that is responsible for producing this result.

The listing below is an excerpt of an OWL document produced by the search of publications with "FrameWeb" in their names. Publications that are returned by the applications are placed under the `<results>` tag, while objects associated with them are placed under `<instancesList>` tag. The association is made using the UUID of each object.

```
<results>
	<instance>
		<uuid>2a6304f5-34c9-4356-a1ce-baa1e7b99e04</uuid>
		<areasOfInterest>130c2f70-5a37-4ad3-815b-841922584cd9</areasOfInterest>
		<areasOfInterest>93fcbf36-cdfe-4fd3-be23-a0d6ab3b45e8</areasOfInterest>
		<publishDate>2007-06-20</publishDate>
		<summary>An Application of the S-FrameWeb Method</summary>
		<participants>e5242491-b2be-4a34-b7b4-b0d9b7537517</participants>
	</instance>
</results>
<instancesList>
	<instance>
		<uuid>130c2f70-5a37-4ad3-815b-841922584cd9</uuid>
		<name>Semantic Web</name>
	</instance>
	<instance>
		<uuid>93fcbf36-cdfe-4fd3-be23-a0d6ab3b45e8</uuid>
		<name>WebApps</name>
	</instance>
	<instance>
		<uuid>e5242491-b2be-4a34-b7b4-b0d9b7537517</uuid>
		<name>Vítor Souza</name>
		<birthDate>1981-06-15</birthDate>
		<gender>M</gender>
		<profession>Professor</profession>
		<institution>UFES</institution>
		<type>2e9c5b6e-0d0f-4da0-b99b-24178ca6873a</type>
	</instance>
</instancesList>
```

Since this result should be based on the application ontology, it was necessary to use an ontology parser. For this purpose, we chose the Jena Ontology API, a framework that provides a programmatic environment for many ontology languages, including OWL. With Jena and Java's reflection mechanisms, the OWL Result Class reads all properties that are made available to the Web page by the action, produces an OWL document containing their information and delivers it to the software agent.

Testing should be conducted in order to check not only the source code, but also the ontologies codified in OWL. In the context of S-FrameWeb, however, this is still open to research and study. Deployment works as the same as other WISs, but should also include the OWL files in a specific place in order to be used by the Front Controller's extension.

5. Future Trends

Web Engineering is a relatively new field of research. New methods, languages and frameworks are proposed to provide practitioners with tools that can facilitate and increase the productivity when developing WebApps.

FrameWeb is a new tool, targeting WISs that have their architecture based on frameworks. By suggesting a standard architecture and bringing concepts from the frameworks to the design models, developers can translate models to code more easily and designer have more control on the outcome of the implementation.

FrameWeb was first applied in the development of the Portal of the Software Engineering Lab – LabES. First, developers were trained in general concepts of Web Engineering, in 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. However, 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.

Two other case studies were conducted. The local Java User Group ESJUG[22] modeled a collaborative learning environment called JSchool[23] using FrameWeb for the same set of frameworks used in the LabES Portal project. This helped mature the method in its initial version.

Another case study reimplemented the LabES Portal changing the Front Controller framework. This helped identify some extensions that should be added to FrameWeb in order to cope with some characteristics of different frameworks. For instance, this work suggested the addition of the `<<formBean>>` stereotype for the navigation model to represent how the framework Struts sends data from the web page to the action class. It also reached the conclusion that the navigation model in FrameWeb is somewhat dependent on the instance of Front Controller frameworks used, and not generic as it was assumed before.

More case studies should be conducted to assess the effectiveness of the method and its appropriateness to different instances of frameworks. Many improvements can come from more practical experiences.

The use of framework-based architectures is becoming the standard for implementation of medium-to-large-sized WIS. Taking the Java platform as example, the definition of standards as JavaServer Faces (JSF)[24] for Web development and the new Enterprise JavaBeans (version 3.0)[25] 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.

The research on the Semantic Web points out to the future of the World Wide Web. Methods for the development of WISs should prepare for, or even help build, this new paradigm. S-FrameWeb suggests a software process that facilitates the development of Semantic WISs by automating

[22] http://esjug.dev.java.net
[23] http://jschool.dev.java.net
[24] http://jcp.org/en/jsr/detail?id=127
[25] http://jcp.org/en/jsr/detail?id=220

certain tasks concerning the generation of semantic annotations on dynamic Web pages.Nonetheless, FrameWeb and S-FrameWeb are far from ideal: there are several opportunities to improve the method. Future work may include:

- Further research on the impact of the use of frameworks and FrameWeb on the activity of Testing. The current work provides no discussion on the subject of testing;
- Proposals on layout and interaction models. Complete methods for the design of WebApps should include models that model aesthetics and usability;
- Conduction of more formal experiments with the method, evaluating more precisely the gains in the productivity of the development team. Currently, only informal experiments have been conducted and conclusions have been reached by requesting developer's opinions;
- 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;
- To make FrameWeb's models more generic, the development of an ontology on Web Applications and frameworks to guide the evolution of FrameWeb's modeling language. New concepts brought by new frameworks could be included in the ontology and, thus, taken to the modeling language;
- Continuation on the research on the Semantic Web and in-practice experiments on the construction of a Semantic WIS using S-FrameWeb;
- Deeper discussions on how to tackle specific Semantic Web issues such as: how will agents find the desired web page?, how will they know how to interact with it?, how will they know if a concept "table" refers to a piece of furniture or a systematic arrangement of data usually in rows and columns?, will a top-level ontology be used for all the Internet?
- Evaluation on how to use Semantic Web Services with S-FrameWeb instead of the dynamic page approach and a comparison of both solutions.

6. Conclusions

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.

Parallel to this, many frameworks and containers for the implementation of WISs were created, denoting the need for a basic infra-structure that helps on the quick development of reliable software with low future maintenance costs. With several ready-to-use and extensively tested components, frameworks promote reuse and good programming practices.

The large utilization of these frameworks and containers by practitioners and the absence of a design method based directed to them has motivated the proposal of FrameWeb, a method based on frameworks for the design of WISs. The current research on the Semantic Web, with many efforts on bringing this idea to reality has impelled us to extend this method and create S-FrameWeb: a method based in frameworks to the construction of semantic WISs.

Given all of the options available, FrameWeb comes in as another one that targets a specific architecture, one based on 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.

S-FrameWeb complements FrameWeb, adding activities that promote the construction of Semantic WISs. Given that the Semantic Web vision will not come true unless Web authors add semantic to their websites, S-FrameWeb is a step in that direction, giving directives for WISs developers to follow in order to add Semantic to Web Applications.

7. References

Ahmad, R., Li, Z., & Azam, F. (2005). Web Engineering: A New Emerging Discipline. In *Proceedings of the IEEE Symposium on Emerging Technologies* (pp.445-450). Catania, Italy: IEEE.

Alur, D., Malks, D., & Crupi, J. (2003). *Core J2EE Patterns: Best Practices and Design Strategies, 2nd edition*. Prentice Hall.

Ambler, S., & Jeffries, R. (2002). *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process, 1st edition*. John Wiley & Sons.

Arch-Int, S., Batanov, D. N. (2003). Development of industrial information systems on the Web using business components. *Computers in Industry*, 50 (2), 231-250, Elsevier.

Bauer, C., & King, G. (2004). *Hibernate in Action, 1st edition*. Manning.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, 284, 34-43.

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide. 2nd edition*. Addison-Wesley Professional.

Ceri, S., Fraternali, P., & Bongio, A. (2000). Web Modeling Language (WebML): a modeling language for designing Web sites. *Computer Networks*, 33 (1-6), 137-157, Elsevier.

Conallen, J. (2002). *Building Web Applications with UML, 2nd edition*. Addison-Wesley.

Díaz, P., Montero, S., & Aedo, I. (2004). Modelling hypermedia and web applications: the Ariadne Development Method. *Information Systems*, 30 (8), 649-673, Elsevier.

Đurić, D. (2004). MDA-based Ontology Infrastructure. *Computer Science and Information Systems* 1 (1), ComSIS Consortium.

Falbo R. A., Guizzardi, G., & Duarte, K. C. (2002). An Ontological Approach to Domain Engineering. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering* (pp. 351- 358). Ischia, Italy: Springer.

Falbo, R. A. (2004) Experiences in Using a Method for Building Domain Ontologies. In *Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering* (pp. 474-477), Banff, Alberta, Canadá.

Fons, J., Valderas, P., Ruiz, M., Rojas, G., & Pastor, O. (2003). OOWS: A Method to Develop Web Applications from Web-Oriented Conceptual Models. In *Proceedings of the International Workshop on Web Oriented Software Technology* (pp. 65-70), Oviedo, Spain.

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley.

Fowler, M. (2007). *Inversion of Control Containers and the Dependency Injection pattern*. Retrieved on November 19, 2001, from http://www.martinfowler.com/articles/injection.html

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Ginige, A., & Murugesan, S. (2001). Web Engineering: An Introduction. *IEEE Multimedia*, 8 (1), 14-18, IEEE.

Gomez-Perez, A., Corcho, O., & Fernandez-Lopez, M. (2005). *Ontological Engineering*. Springer.

Guarino, N. (1998). Formal Ontology and Information Systems. In *Proceedings of the 1st International Conference on Formal Ontologies in Information Systems* (pp. 3-15), Trento, Italy: IOS Press.

Jacobson I., Booch G. & Rumbaugh J. (1999). *The Unified Software Development Process*, Addison Wesley.

Jacyntho, M. D., Schwabe, D., & Rossi, G. (2002). A Software Architecture for Structuring Complex Web Applications. In *Proceedings of the 11th International World Wide Web Conference, Web Engineering Alternate Track*, Honolulu, EUA: ACM Press.

Koch, N., Baumeister, H., Hennicker, R., & Mandel, L. (2000). Extending UML to Model Navigation and Presentation in Web Applications. In *Proceedings of Modelling Web Applications in the UML Workshop*, York, UK.

Koch, N., & Kraus, A. (2002). The Expressive Power of UML-based Web Engineering. In D. Schwabe, O. Pastor, G. Rossi e L. Olsina (Ed.), *Proceedings of the Second International Workshop on Web-Oriented Software Technology* (pp. 105-119), CYTED.

Krutchen, P. (2000). *The Rational Unified Process: An Introduction, 2nd edition*. Addison-Wesley.

Lee, S. C., & Shirani, A. I. (2004). *A component based methodology for Web application development*. *Journal of Systems and Software*, 71 (1-2), 177-187, Elsevier.

Lima, F., & Schwabe, D. (2003). Application Modeling for the Semantic Web. In Proceedings of the 1st Latin American Web Conference (pp. 93-102), Santiago, Chile: IEEE-CS Press.

McIlraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic Web Services. *Intelligent Systems*, 16 (2), 46-53, IEEE.

Murugesan, S., Deshpande, Y., Hansen, S., & Ginige, A. (1999). Web Engineering: A New Discipline for Development of Web-based Systems. In *Proceedings of the 1st ICSE Workshop on Web Engineering* (pp. 3-13). Australia: Springer.

Narayanan, S., & McIlraith, S. A. (2002). Simulation, Verification and Automated Composition of Web Services. In *Proceedings of the 11th international conference on World Wide Web* (pp. 77-88), Hawaii, USA: ACM.

Neighbors, J. M. (1981). *Software Construction Using Components*. Ph.D. Thesis. Department of Information and Computer Science, University of California, Irvine.

OMG (2007). *Ontology Definition Metamodel Specification*. Retrieved on January 29, 2007, from http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf

Pastor O., Gómez, J., Insfrán, E., & Pelechano, V. (2001). The OO-Method Approach for Information Systems Modelling: From Objetct-Oriented Conceptual Modelling to Automated Programming. *Information Systems*, 26 (7), 507-534, Elsevier.

Pressman, R. S. (2005). *Software Engineering: A Practitioner's Approach, 6th edition*. McGraw Hill.

Reenskaug, T. (1979). *THING-MODEL-VIEW-EDITOR, an Example from a planning system*. Xerox PARC Technical Note.

Resende, A., & Silva, C. (2005). *ProgramaÁ„o Orientada a Aspectos em Java*. Brasport.

Rosenberg, D., Scott, K. (1999). *Use Case Driven Object Modeling with UML : A Practical Approach*. Addison-Wesley.

Schmidt, D. (2007). *"Programming Principles in Java: Architectures and Interfaces", chapter 9*. Retrieved on January 29, 2007 from http://www.cis.ksu.edu/~schmidt/CIS200/

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

Shannon, B. (2003). *Java™ 2 Platform Enterprise Edition Specification, v1.4*. Sun Microsystems.

Souza, V. E. S., & Falbo, R. A. (2007). FrameWeb - A Framework-based Design Method for Web Engineering. In *Proceedings of the Euro American Conference on Telematics and Information Systems 2007* (pp. 17-24). Faro, Portugal: ACM Press.

Souza, V. E. S., Lourenço, T. W., Falbo, R. A., & Guizzardi, G. (2007). S-FrameWeb – a Framework-based Design Method for Web Engineering with Semantic Web Support. In *Proceedings of Workshops and Doctoral Consortium of the 19th International Conference on Advanced Information Systems Engineering* (pp. 767-778), Trondheim, Norway.

Standing, C. (2002). Methodologies for developing Web applications. *Information and Software Technology*, 44 (3), 151-159, Elsevier.

W3C (2007a). *OWL Web Ontology Language Guide, fev. 2004*. Retrieved on December 17, 2007, from http://www.w3.org/TR/owl-guide/

W3C (2007b). *W3C Glossary and Dictionary*. Retrieved on January 23, 2007, from http://www.w3.org/2003/glossary/