S-FrameWeb: a Framework-Based Design Method for Web Engineering with Semantic Web Support

Vítor Estêvão Silva Souza[1], Thiago Wotikoski Lourenço[1],
Ricardo de Almeida Falbo[1], Giancarlo Guizzardi[1,2]

[1] Universidade Federal do Espírito Santo, Av. Fernando Ferrari, 514
29075-910 Vitória – ES, Brazil
[2] Laboratory for Applied Ontology, Polo Tecnologico, Via Solteri, 38
38100 Trento, Italy
{vitorsouza, twotikoski}@gmail.com, falbo@inf.ufes.br, guizzardi@loa-cnr.it

Abstract. The Web Engineering area is evolving fast. Many methods and frameworks to support Web Information Systems (WISs) development have already been proposed. Particularly, the use of frameworks and container-based architectures is state-of-the-practice. Motivated by this scenario, we have proposed a method for designing framework-based WISs called FrameWeb. However, we should consider that the Semantic Web has been gaining momentum in the last few years. The idea is that the information on the Web should be available in machine-processable formats so that software agents could reason with it. This paper presents an extension to FrameWeb, called S-FrameWeb, that aims to support the development of Semantic WISs.

Keywords: Web Engineering, Web Information Systems, Frameworks, Semantic Web.

1 Introduction

The Semantic Web is being considered the future of the World Wide Web (WWW). Coined by Berners-Lee [1], 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 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.

Methods already exist for the development of Web Information Systems (WISs), such as WAE [2], OOWS [3] and OOHDM [4]. In this context, we proposed a method for the design of WISs that are based on frameworks, called FrameWeb (Framework-Based Design Method for Web Engineering) [5]. FrameWeb proposes a basic

architecture for developing WISs, a set of activities and a UML profile for a set of design models that brings concepts used by some categories of frameworks. The idea is that the use of FrameWeb would further improve team productivity by using a modeling language that would allow designers to produce diagrams that represent framework concepts, and developers (maybe, in the future, CASE tools) to directly translate these diagrams to code [5].

To help developers build WISs with semantic annotations, we decided to work on an extension of FrameWeb, called S-FrameWeb. The idea is to 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.

This paper presents S-FrameWeb, and it is organized as follows: section 2 discusses some issues concerning WebE and the Semantic Web and briefly presents FrameWeb. Section 3 presents S-FrameWeb and how it proposes to build "Semantic Web-enabled" applications. Section 4 discusses related work. Finally, section 5 presents our conclusions and potential future work.

2 Web Engineering and the Semantic Web

Web Engineering (WebE) has been defined as "the establishment and use of engineering principles and disciplined approaches to the development, deployment and maintenance of Web-based Applications" [6]. WebE was conceived at a time when Web Applications (WebApps) were developed in an ad-hoc manner, without a methodology or software process to support developers. Nowadays, however, there are many methods, such as WAE [2], OOWS [3] and OOHDM [4], that are being used.

Also, technologies for codifying WebApps have evolved. The use of frameworks to support the construction of complex Web Information Systems (WISs) is state-of-the-practice. Container-based architectures, such as the most recent version of the Java Enterprise Edition [7] standard, also borrow many concepts from these frameworks. Both frameworks and container-based architectures promote the reuse of a commonly used application infrastructure and improve productivity.

There are many different frameworks available for coding WISs. However, it is possible to separate them into few categories organized by purpose [5]. Table 1 lists four of these categories: Front Controller [8], Decorator, Object/Relational (O/R) Mapping [9] and Dependency Injection frameworks [10]. Other kinds of frameworks include: Aspect-Oriented Programming frameworks, Authentication & Authorization frameworks, Search engines, etc.

Table 1. Frameworks that form a commonly used infrastructure for Web Applications.

Framework	Purpose
Front Controller	Also known as MVC frameworks, defines an architecture that separates the functionality of a WebApp from its presentation based on the Model-View-Controller pattern [11].

Framework	Purpose
Decorator	Based on the Decorator design pattern [11], automates the task of making every web page of the site have the same layout (header, footer, navigation bar, colors, images, etc).
Object/Relational (O/R) Mapping	Provides automatic and transparent persistence of objects to tables of a RDBMS using meta-data that describe the mapping between both worlds [9].
Dependency Injection	Allows the developer to program to interfaces [10] and specify the concrete dependencies in a configuration file. The idea is that classes that depend on services from different tiers would declare an association with an interface instead of the concrete implementation. This facilitates, for instance, the replacement of the real service class with a mock object for unit testing.

These frameworks can substantially change the architecture and the components that must be developed for a WIS. That motivated the proposition of the Framework-based Design Method for Web Engineering (FrameWeb). The interested reader should refer to [5] and [12] for detailed information. FrameWeb proposes:

- A standard architecture for Web Applications that integrates with those frameworks by separating their concerns into different packages;
- A UML profile suited for the construction of four kinds of design models that represent framework components from different packages: Domain Model, Persistence Model, Navigation Model and Application Model [12];
- Although FrameWeb does not prescribe a software process, allowing organizations to use the process that suits them best, it suggests that use cases and class diagrams are used during requirement analysis.

FrameWeb's standard architecture divides the system into three main tiers, as shown in Figure 1. The Presentation Logic tier contains elements related to Web-based user interfaces. The **`Controller`** package gathers the action classes that integrate with the Front Controller framework, while the **`View`** package contains Web pages, style sheets, images and other files related with the exhibition of information.

The Business Logic tier also contains two packages: **`Domain`** and **`Application`**. The former includes classes that represent domain concepts modeled during requirement analysis. The latter comprises classes that implement functionalities represented as use cases during that same stage.

The last tier regards Data Access. The **`Persistence`** package contains classes that communicate with the Object/Relational (O/R) Mapping framework to create, retrieve, update and delete domain objects from the persistence store. FrameWeb suggest the use of the Data Access Object pattern [8] for this package.

The dependency associations in figure 1 show how these packages interact. User stimuli come from **`View`** components and reach the **`Controller`** classes by means of the MVC framework. The action classes in `Controller` call methods from **`Application`** classes, which manipulate **`Domain`** objects and also depends on the

`Persistence` of these objects. Associations stereotyped as `<<weak>>` represent loose coupling. For instance, the packages in the Presentation tier do not create and manipulate domain objects directly, but use them to display data or pass them around as parameters, using a domain-driven approach[1].

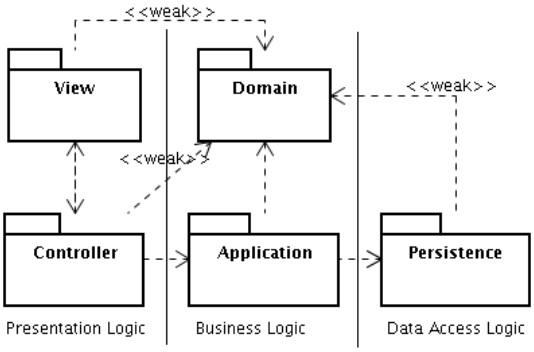

Fig. 1. FrameWeb's standard architecture for WIS [5].

To model classes and other components that belong to the different packages of the standard architecture, FrameWeb uses UML's lightweight extension mechanism to create a profile for designing four different kinds of diagrams [12] during system design, which are summarized in Table 1. All of them are based on UML's class diagram, but represent components from different packages that integrate with different frameworks. Interested readers should refer to [12] for further details.

Table 2. Diagrams built during the design of a WIS using FrameWeb.

Diagram	Purpose
Domain Model	Represents domain classes modeled during analysis, complemented with platform-dependent information (attribute types, association navigabilities, etc.) and O/R mappings (which are more easily represented in this model instead of the Persistence Model because the attributes are modeled here). Guides the implementation of classes from the `Domain` package and also the configuration of the O/R framework.
Persistence Model	Shows DAO classes that are responsible for the persistence of domain objects and the existence of specific queries to the database. Every domain class that requires persistence should have a DAO interface and an implementation for each persistence technology used. Guides the codification of the DAOs, which belong to the `Persistence` package, and the creation of specific database queries on the O/R framework.

1 In this context, the domain-driven approach (referred to as model-driven approach by the framework's documentation) consists of using an instance of a domain class as wrapper for its attributes when they are passed as parameters.

Diagram	Purpose
Navigation Model	Displays components from the presentation tier, such as web pages, HTML forms, templates, binary files and action classes, and their relationships among themselves. Guides the implementation of action classes (`Controller` package), other view components (`View` package) and the configuration of the Front Controller framework.
Application Model	Models the interfaces and classes that implement use case functionalities and the dependency chain from the action classes (which depend on them) until the DAOs (which they depend on). Guides the codification of classes from the `Application` package and the configuration of the Dependency Injection framework.

FrameWeb provides a way for modeling WIS that is suited for those based on frameworks. There is no indication, however, on how to provide semantic annotations that could make the WIS available for Semantic Web agents to reason with it. Reasoning means that software agents are able to understand the information presented by Web pages and take sensible actions according to a goal that was previously given. The most usual way for agents to understand the contents of a website is by semantically 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 [13]. Along with ontology representation languages such as OWL [14], 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, establish relation between terms, systematic classification and infer information from explicitly defined information in this structure.

If the ontology is built (using one of many methodologies for their construction [15]) and given the availability of tools such as OILEd[2] and Protégé[3], the annotation of static Web pages with OWL has become a straightforward task.

However, few websites are composed strictly by static pages. What is commonly seen is Web pages being dynamically generated by software retrieving information on-the-fly from data repositories such as relational databases. These data-intensive websites have the advantage of separating data and layout, but also have limitations such as being invisible to search engines and not being comprehensible by software agents [16]. Thus, an automated way of annotating dynamic Web pages is needed.

One way to do that is, when a Web page is requested at the Web server, it must recognize if the request comes from a human agent or a software agent. In the latter case, instead of generating a HTML human-readable Web page, the server should return a document written in an ontology specification language (e.g. OWL) containing meta-data about the information that would be conveyed in the page.

2 http://oiled.man.ac.uk/

3 http://protege.stanford.edu/

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?

Hepp [17] advocates that semantic annotation of static or dynamic data is not enough and that the original vision of the Semantic Web can only be achieved by the utilization of Semantic Web Services. A Web Service is "a software system designed to support interoperable machine-to-machine interaction over a network" [18]. Web Services provide a nice way for software agents to interact with other systems, requesting services and processing their results.

Many researchs are now directed to the use of Web Services on the Semantic Web. Semantic Web Services are formed by adding semantic annotations to Web Services so they 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 [19]. Many tasks are expected to be automated with this, including service discovery, invocation, interoperation, selection, composition and monitoring [20].

3 Semantic FrameWeb

The main goal of S-FrameWeb is to make WISs "Semantic Web-enabled". This should be accomplished by the Front Controller framework, which identifies 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 fulfill its purpose, S-FrameWeb adds three new steps to FrameWeb's software process: domain analysis, ontology design and ontology implementation. A suggested software process is shown in figure 2. These steps are further discussed next.

3.1 Domain Analysis

To bring a WIS to the Semantic Web it is imperative to formally describe its domain. As stated in section 2, the most usual way of doing this is by constructing 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).

Domain Analysis is "the activity of identifying the objects and operations of a class of similar systems in a particular problem domain" [21, 22]. 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 [22].

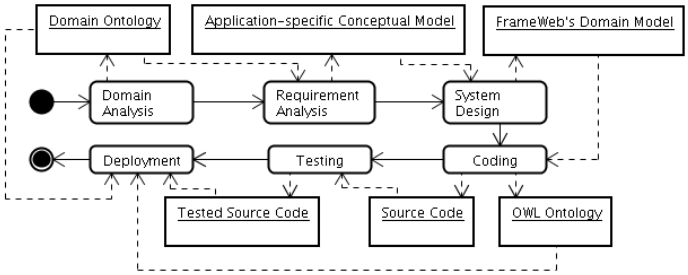

Fig. 2. The software process suggested by S-FrameWeb.

For a diagrammatic representation of the ontology, S-FrameWeb uses OMG's[4] Ontology Definition Metamodel (ODM) [23], "a language for modeling Semantic Web ontologies in the context of MDA" [24]. ODM defines an ontology UML profile that allows developers to represent ontologies in UML class diagrams.

The output of Domain Analysis is an ontology that represents concepts from the problem domain. The ontology's diagram can be reused in the Requirement Analysis phase to produce the application's conceptual model, which will later be refined and become FrameWeb's Domain Model (FDM) during system design.

Table 3 summarizes the evolution of the models throughout the software process.

Table 3. Models produced by the software process suggested by S-FrameWeb

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.

Figure 3 shows the conceptual model for a very simple culinary recipes WIS called "Cookbook". This application includes the registry of recipes and a simple search feature. After the domain of culinary was analyzed and an ontology modeled, the conceptual model was built in ODM using only the classes that were required for this particular application.

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.

4 Object Management Group – http://www.omg.org/ontology/

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. More on ODM's semantics can be found at [23].

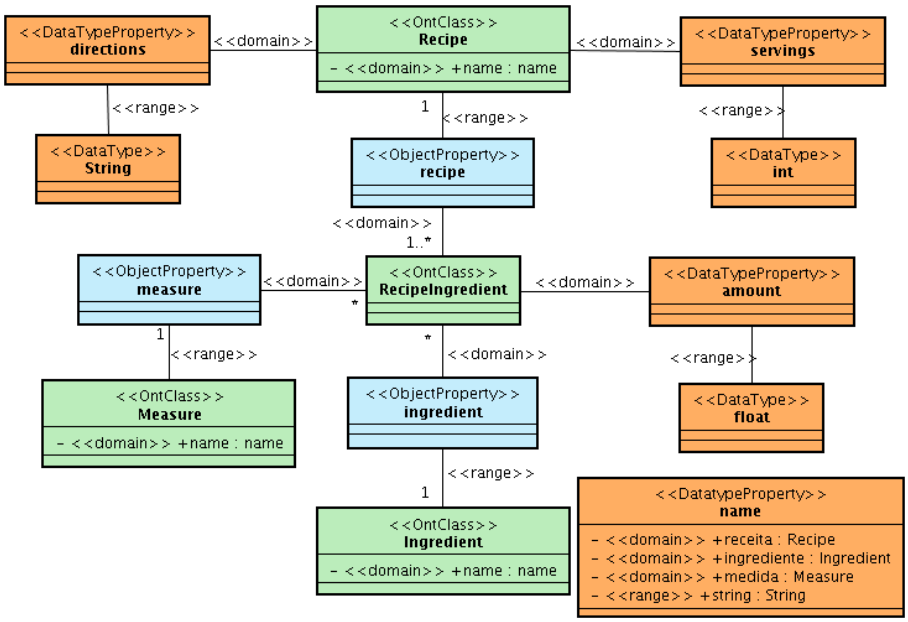

Fig. 3. The conceptual model for the Cookbook application.

3.2 Ontology Design

During the design phase of the software process, FrameWeb prescribes the construction of a Domain Model (referred to as FDM), which shows classes that represent concepts from the problem domain and their object/relational mappings.

S-FrameWeb proposes an extension to this diagram, mixing both FrameWeb's and ODM's UML profiles to build FDM. Based on the conceptual model, the designer should simplify ODM's syntax and add the OR mappings. Figure 4 shows the FDM for the Cookbook application. We can see that **`<<DataType>>`** elements were replaced by class attributes (this should happen only in simple cases – for instance, when they do not participate in a subsumption hierarchy or when the datatype is not structured) and that some mappings were included (`{not null}` and `{cascade=all}` [12]).

3.3 Ontology Implementation

Finally, at the coding phase, the domain ontology and the FDM should be coded in OWL and deployed into the WIS. In this context, “deploy” means placing the OWL

file in a predetermined location so the Front Controller framework can read it. Because the models are represented in ODM, their codification in OWL are straightforward (ODM proposes graphical representations for OWL constructs) and, in the near future, probably this can be automated by CASE tools.

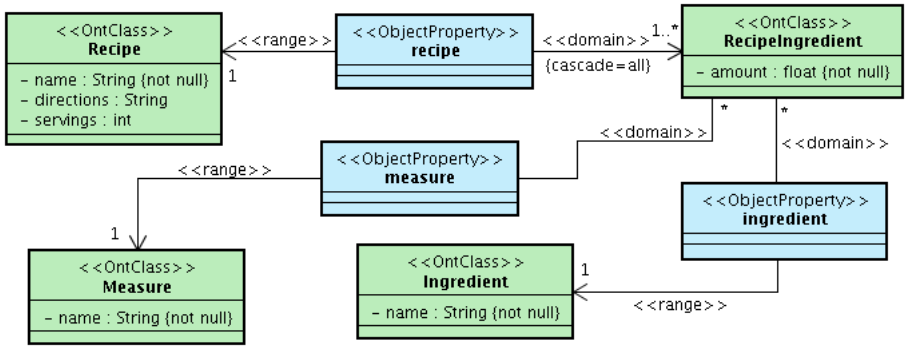

Fig. 4. S-FrameWeb's Domain Model for the Cookbook application.

During the execution of the WIS, the Front Controller provides an infrastructure that identifies when the request comes from a software agent, reads both ontology's and FDM's OWL files and responds to the request based on the execution of an action and reasoning over the ontologies. Since existing Front Controller frameworks do not have this infrastructure, a prototype of it was developed and is detailed next.

3.4 Front Controller Infrastructure

To experiment S-FrameWeb in practice, we have extended the Struts[2] framework[5] to recognize software agents requests and respond with an OWL document, containing the same information it would be returned by that page, but codified as OWL instances. Together with the OWL files for the domain ontology and the FDM, software agents can reason about the data that resulted from the request.

Figure 5 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. This is an expected behavior of Struts[2] and most of the framework's features are implemented as interceptors (e.g., to have it manage a file upload, use the `fileUpload` interceptor).

An “OWL Interceptor” was developed and configured as first interceptor of the stack. When the request is passing through the stack, the OWL Interceptor verifies if a specific parameter was sent by the agent in the request (e.g. `owl=true`), indicating that the action should return an “OWL Result”. If so, it creates a pre-result listener that will deviate successful requests to another custom-made component that is responsible for producing this result, which we call the “OWL Result Class”. Since

5 http://struts.apache.org/2.x/

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.

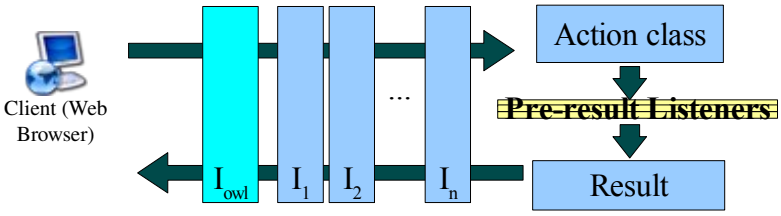

Fig. 5. S-FrameWeb's Front Controller framework extension for the Semantic Web.

Using Jena and Java's reflection API, the OWL Result Class obtains all accessor methods (JavaBeans-standardized get*Property*() methods) of the Action class that return a domain object or a collection of domain objects. These methods represent domain information that is available to the client (they are called by the result page to display information to the user on human-readable interfaces). They are called and their result is translated into OWL instances, which are returned to the client in the form of an OWL document.

4 Related Work

As the acceptance of the Semantic Web idea grows, more methods for the development of "Semantic Web-enabled" WebApps are proposed.

The approach which is more in-line with our objectives is the Semantic Hypermedia Design Method (SHDM) [25]. SHDM is a model-driven approach for the design of Semantic WebApps based on OOHDM [4]. SHDM proposes five steps: Requirement Gathering, Conceptual Design, Navigational Design, Abstract Interface Design and Implementation.

Requirements are gathered in the form of scenarios, user interaction diagrams and design patterns. The next phase produces a UML-based conceptual model, which is enriched with navigational constructs in the following step. The last two steps concern interface design and codification, respectively.

SHDM is a comprehensive approach, integrating the conceptual model with the data storage and user-defined templates at the implementation level to provide a model-driven solution to WebApps development. Being model-driven facilitates the task of displaying agent-oriented information, since the conceptual model is easily represented in OWL.

While the approach is very well suited to content-based WebApps, WIS which are more centered in providing functionalities (services) are not as well represented by SHDM. The proposal of FrameWeb was strongly motivated by the current scenario where developers are more and more choosing framework or container-based

architectures to create applications for the Web. S-FrameWeb builds on top of FrameWeb to provide semantics to these functionality-based WebApps.

5 Conclusions and Future Work

S-FrameWeb suggests a software process that facilitates the development of Semantic WISs by automating certain tasks concerning the generation of semantic annotations on dynamic Web pages. However, the following limitations have already been identified and are bound to future work:

- Software agents must know how to find the Web pages. Pages that are linked by others can be found by search engines, but that is not the case with the ones that represent the request for a service. The research on Web Service discovery could provide some insight on this issue;
- Agents must speak a common language to understand Web pages. If an instance of "table" is returned, how will the agent know if it's a "piece of furniture", "a systematic arrangement of data usually in rows and columns"[6] or any other meaning? The use of top-level ontologies, such as Dolce[7], should be considered for this matter;
- Works in the area of Semantic Web Services [19, 20, 26] suggest another way to deal with the issue of annotation of WISs. S-FrameWeb should be implemented to use Web Services in the future to compare both approaches;
- The infrastructure prototype was developed for the Struts[2] framework and currently has some limitations that have to be addressed. Other frameworks should be extended so S-FrameWeb can be used in different platforms.

Acknowledgments. This work was accomplished with the financial aid of CAPES, an entity of the Brazilian Gov't dedicated to scientific and technological development.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001) n. 284, p. 34-43
2. Conallen, J.: Building Web Applications with UML. 2nd edn. Addison-Wesley (2002)
3. Fons, J.; Valderas, P.; Ruiz, M.; Rojas, G.; Pastor, O: OOWS: A Method to Develop Web Applications from Web-Oriented Conceptual Models. Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI), Orlando, USA (2003)
4. Schwabe, D., Rossi, G.: An Object Oriented Approach to Web-Based Application Design. Theory and Practice of Object Systems 4 (4). Wiley and Sons (1998)
5. Souza, V. E. S., Falbo, R. A.: FrameWeb: A Framework-based Design Method for Web Engineering. Proceedings of the Euro American Conference on Telematics and Information Systems, Faro, Algarve, Portugal (2007)

6 Merriam-Webster Online Dictionary (http://www.m-w.com)

7 More about Dolce at http://wonderweb.semanticweb.org/deliverables/documents/D18.pdf

6. Murugesan, S., Deshpande, Y., Hansen, S., Ginige, A.: Web Engineering: A New Discipline for Development of Web-based Systems. Proceedings of the First ICSE Workshop on Web Engineering. IEEE, Australia (1999)
7. Shannon, B.: Java™ Platform, Enterprise Edition (Java EE) Specification, v5. Sun Microsystems (2006)
8. Alur, D., Crupi, J., Malks, D.: Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall / Sun Microsystems Press (2001)
9. Bauer, C., King, G.: Hibernate in Action. 1st edn. Manning (2004)
10. Fowler, M.: Inversion of Control Containers and the Dependency Injection Pattern (http://www.martinfowler.com/articles/injection.html). Captured on July 19th (2006)
11. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994)
12. Souza, V. E. S., Falbo, R. A.: A Language for Modeling Framework-based Web Information Systems. Proceedings of the 12th International Workshop on Exploring Modeling Methods in System Analysis and Design. Trondheim, Norway (2007)
13. Guarino, N.: Formal Ontology and Information Systems. Proceedings of the 1st International Conference on Formal Ontologies in Information Systems. IOS Press. Trento, Italy (1998) p. 3-15.
14. W3C: OWL Web Ontology Language Guide, fev. 2004 (http://www.w3.org/TR/owl-guide/). Captured on: November 13th (2006)
15. Gomez-Perez, A., Corcho, O., Fernandez-Lopez, M.: Ontological Engineering. Springer (2005)
16. Stojanovic, L., Stojanovic, N., Volz, R.: Migrating data-intensive Web Sites into the Semantic Web. Proceedings of the 2002 ACM symposium on Applied computing. ACM. Madrid, Spain (2002) p. 1100-1107
17. Hepp, M.: Semantic Web and semantic Web services - Father and Son or Indivisible Twins? IEEE Internet Computing. IEEE (2006) v. 10, n. 2, p. 85-88
18. W3C: W3C Glossary and Dictionary (http://www.w3.org/2003/glossary/). Captured on: January 23rd (2007)
19. McIlraith, S. A., Son, T. C., Zeng, H.: Semantic Web Services. Intelligent Systems. IEEE (2001) v. 16, n. 2, p. 46-53
20. Narayanan, S., McIlraith, S. A.: Simulation, Verification and Automated Composition of Web Services. Proceedings of the 11th international conference on World Wide Web. ACM. Hawaii, USA (2002) p. 77-88
21. Neighbors, J. M.: Software Construction Using Components. Ph.D. Thesis. Department of Information and Computer Science, University of California, Irvine (1981)
22. Falbo R. A., Guizzardi, G., Duarte, K. C. : An Ontological Approach to Domain Engineering. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'2002). Ischia, Italy (2002). pp. 351- 358
23. OMG: Ontology Definition Metamodel Specification (http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf). Captured on: January 29th (2007)
24. Đurić, D.: MDA-based Ontology Infrastructure. Computer Science and Information Systems. ComSIS Consortium (2004) vol. 1, issue 1
25. Lima, F., Schwabe, D.: Application Modeling for the Semantic Web. First Latin American Web Conference (LA-Web). IEEE-CS Press. Santiago, Chile (2003)
26. Trastour, D., Bartolini, C., Preist, C.: Semantic Web Support for the Business-to-Business E-Commerce Lifecycle. Proceedings of the 11th International World Wide Web Conference (WWW 2002). Hawaii, USA (2002)