A Tool for Efficient Development of Ontology-based Applications

Olavo Holanda[1], Ig Ibert Bittencourt[1], Seiji Isotani[2], Endhe Elias[1], Judson Bandeira[1]

[1]Computing Institute – Federal University Alagoas (UFAL)
Maceió – AL – Brazil

[2]Department of Computer Science, Institute of Mathematics and Computer Science
University of São Paulo, São Carlos, São Paulo, Brazil

{olavo.holanda,ig.ibert, endhe.elias, jmb}@ic.ufal.br, sisotani@icmc.usp.br

Abstract. *In the past few years, the use of ontologies for creating more intelligent and effective application has increased considerably. This growth is due to the fact that ontologies attempt to provide semantics to the data consumed by machines, so that they can reason about these data. However, the development of applications based on ontologies is still difficult and time-consuming, because the existing tools lack to provide a simple and unified environment for the developers. Most of these tools only provide data manipulation using RDF triples, complicating the development of applications that need to work with the object orientation paradigm. Furthermore, tools which provide instances manipulation via object orientation do not support features such as manipulating ontologies, reasoning over rules or querying data with SPARQL. In this context, this work proposes a tool for supporting the efficient development of ontologies-based applications through the integration of existing technologies and techniques. In order to evaluate the benefits of this tool, a controlled experiment with eight developers (unfamiliar with ontologies) was performed comparing the proposed tool with another one used by the community.*

1. Introduction

Recently, ontologies had obtained quite attention in the computer scientific community. The term, which has origins in philosophy, becomes a useful word in computer science for a new approach of knowledge representation about real-world entities. For this, ontologies offer a shared understanding of a particular domain and a formalization which allows its data to be interpretable by machines[Hepp et al. 2007].

As a result, ontologies are not only applied as basis for the so called Semantic Web, but in other areas of computing research and industry. For example, e-commerce applications use ontologies for parametric searches and heterogeneous systems integration[Das et al. 2002]. Another industry segment is the media systems that has used this approach to do real-time data inference, delivering up-to-date content for its users[Kiryakov et al. 2010]. In addition, several other fields are using ontologies such as medicine [Bard and Rhee 2004], computer mobile [Cheyer and Gruber 2010] and adaptative education [Bittencourt et al. 2009][Bittencourt et al. 2006][Bittencourt and Costa 2011].

This dissemination is a consequence of the growing number of tools and software libraries that allow the development of semantic applications. Currently, more than 170

tools are listed at the semanticweb.org, and this number tends to increase. Despite the high number of tools, not all of them aim to support the development of applications for the Semantic Web. On the other hand, the tools that offer this type of support do not provide it through a simple and unified environment, that is, they fail to offer common functionalities when developing applications based on ontologies, such as managing and querying ontologies, reasoning over rules, manipulating instances via object oriented paradigm (in contrast to the manipulation of instances via triple RDF - *Resource Description Framework*), among others.

In this context, this paper proposes a tool that provides simplified development of ontologies through the object oriented model. Moreover, the tool provides an integration of existing technologies and techniques to create a unified environment for developers of applications based on ontologies. This tool provides services such as operations on ontologies, manipulating instances, SPARQL[1] (*Simple Protocol and RDF Query Language*) queries, data inference over SWRL[2] (*Semantic Web Rule Language*) rules, and so on. This paper is organized as follows. In Section 2, the characteristics of each type of ontology programming are outlined. In Section 3, the proposed tool is described. The performed experiment, evaluating the proposed work is presented in Section 4. Section 5 presents some conclusions and future works.

2. Ontology Programming

During the development of a semantic-based application, the manipulation of instances is one of the steps in the process of development. For this step, there are currently two main approaches used by the ontology management systems: RDF triples and object oriented development. In the next subsections, the main distinctions and benefits of the aforementioned approaches are detailed.

2.1. RDF Triples Development

Most current APIs (*Application Programming Interface*) are still working with the development based on RDF triples (subject, predicate and object). Thus, application developers should be aware of how the ontology works in RDF layer, in order to manipulate the data through each triple in application code.

When the developer desires to add a resource (subject) in the ontology with several properties inherent to it, several lines of code representing each triple of the resource will be necessary. Each triple captures a single value of each property. Similarly, if the application removes this resource, several triples should be removed.

For example, to create an instance *alice* of the entity Person with the datatype property *name* “Alice” using the Sesame API, which works with the development based on RDF triples, several lines of code are needed. Figure 1 shows how to add this resource in the Sesame repository.

[1]SPARQL Query Language for RDF is W3C recommendation since January 2008 - `http://www.w3.org/TR/rdf-sparql-query/`

[2]More information at: `http://www.w3.org/Submission/SWRL/`

```
...
ValueFactory f = myRepository.getValueFactory();

// create some resources and literals to make statements out of
URI alice = f.createURI("http://example.org/people/alice");
URI name = f.createURI("http://example.org/ontology/name");
URI person = f.createURI("http://example.org/ontology/Person");
Literal alicesName = f.createLiteral("Alice");

RepositoryConnection con = myRepository.getConnection();
// alice is a person
con.add(alice, RDF.TYPE, person);
// alice's name is "Alice"
con.add(alice, name, alicesName);
...
```

Figure 1. Code example from Sesame API

2.2. Object Oriented Development

Instead of RDF triples, object-oriented applications manipulate data at object level and their attributes. Such objects are characterized by a set of attributes and values. In this sense, a tool is necessary to "mapp" the operations on objects to the RDF triples infrastructure that works underneath. Some tools were created to provide such paradigm for handling instances in ontologies.

As a result, developers do not need to have a deep knowledge of the ontology representation language. An object in the code represents an instance in the ontology, their attributes are mapped to the properties of the instances and RDF classes become Classes in the programming language. Therefore, to add a resource in the ontology, the developer just need to add the object, facilitating, in this way, the development of such applications.

Comparing to Sesame example, Figure 2 shows the same resource added in the Sesame repository, but now using Alibaba API, which allows the development based on the object-oriented paradigm.

```
...
ObjectConnection con = repository.getConnection();

// create a Person
Person alice = new Person();
alice.setName("Alice");

// add a Person to the repository
con.addObject(alice);
...
```

Figure 2. Code example from Alibaba API

3. Proposed Tool

Currently, there are several tools which manipulate ontology through the paradigm of object orientation (e.g. Jastor [Szekely and Betz 2009] and Elmo [Mika 2007]) instead of RDF triples. However, these tools provide only the manipulation of instances, which is only one step of the ontology-based development process. Therefore, when working with these tools, developers need to search other libraries to build semantic applications with usual features (e.g. query ontologies, handling instances, perform rules, etc.). To alleviate this lack of appropriate development tools to build and maintain semantic applications, this paper proposes a tool (more specifically a Java software library) that integrates several features that facilitate the development based on ontologies.

The system requirements of the proposed tool are:

- **Repositories Handling**: The first service needed for ontology-based development that the tool provides is the remote management of Sesame repositories. This management is composed of creation and delete of repositories in a Sesame server;
- **Persistence of ontologies**: The tool also provides, for the user, ontologies persistence service in repositories. OWL or RDF files can be added to a repository, which stores all data in binary files. Besides adding ontologies, this service allows the delete and retrieve of a given ontology from repositories;
- **Handling Instances**: The main service provided by the tool is the manipulation of ontology instances present in a repository. Note that this manipulation is done through the paradigm of object orientation. This service is composed of the methods for creating, retrieving and removing instances of a repository;
- **Generation of Java code**: To improve the instances manipulation following the paradigm of object orientation, the automatic generation of Java code from ontologies is required. This feature allows developers to "map" ontologies in Java code, creating classes that represent entities and concepts in these ontologies. As a final result of this feature, a Java library is created,
- **SPARQL queries**: When developing any application, it is very common to query the data stored by it. It is not different when the application is based on ontologies, where developers need a service that can query RDF graphs. The proposed tool provides a service which offers queries on ontologies based on SPARQL language;
- **Backup Repository**: Developers often need, for some industry applications, to keep backups of a repository. This feature allows the creation of backup files, recovering these files into an empty repository and the copy between different repositories;
- **Verification and Validation of Ontology**: A service that allows consistency checking of an ontology. In other words, this feature verifies that an ontology does not contain contradictions, i.e., if there is only one interpretation of the concepts in the ontology;
- **Reasoning over Rules**: This service allows the performing of SWRL rules present in the repository, creating new information through the reasoning over them.

3.1. System Architecture

The system architecture is based on the layers pattern [Buschmann et al. 2007], Figure 3, where each layer uses only the services of the layer below. The architecture will be detailed using a bottom-up approach, starting with the layer OWLIM. OWLIM is a set

of semantic repositories of high performance and scalability[Kiryakov et al. 2005]. The proposed tool offers the repository creation with default configuration as OWLIM *Lite* version. The OWLIM was used along with the Sesame server, so that these repositories can be accessed remotely. The integration between Sesame and OWLIM is done by the OWLIM tool, which implements an extension of the Sesame Repository API.

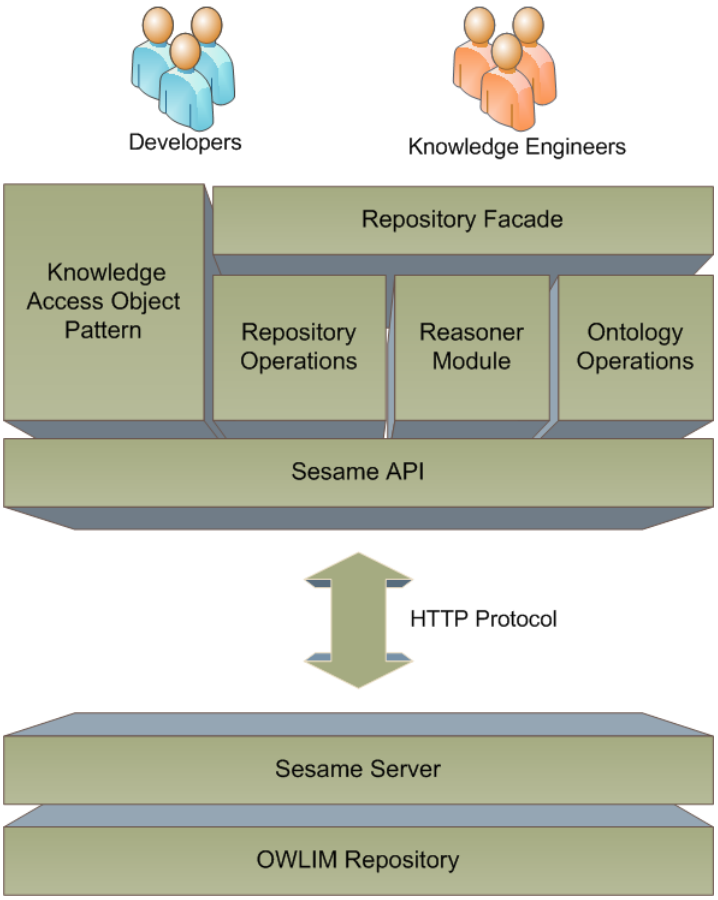

Figure 3. System Architecture

To access a repository in Sesame server, it is necessary to use the Sesame Repository API, which behaves in this case as a client in a client-server architecture [Buschmann et al. 2007]. This API communicates via HTTP (*Hypertext Transfer Protocol*) with Sesame server. In this layer are the methods required to connect to the repository and data manipulation through RDF triples. Above this layer, there are four modules: operations on ontologies, reasoning module, operations on repositories, and the KAO pattern. The layer "operations in ontologies" is composed of functionalities performed on ontologies (ontology itself, not the instances). Among these features are: the insertion and delete of an ontology, generate Java code from one or more OWL files and ontology validation.

On the other hand the layer of "operations in repositories" brings together relevant services for manipulating Sesame repositories. This module allows the developer to operate in remote repositories, such operations can be: the creation or removal of a repository, clear a repository (delete all data present on it) and create backup copies of a given repository.

The next layer to be detailed is the "reasoning module". This module has the goal to infer new data in a repository by executing SWRL rules present on it. Once you run this module, it will look if there are rules in the repository, if any, it will check for new data to be inferred from the execution of those discovered rules. See on Figure 1 that these last three discussed layers are under the "repository facade" layer, which is designed to provide a unified access to these three modules through the Facade design pattern [Gamma et al. 2004].

Last but not least, comes the layer called "Knowledge Access Object" (KAO), which is a persistent pattern similar to the Data Access Object (DAO), with the difference that the KAO not only works with data, but works with information from the ontologies. The KAO pattern aims to provide an abstraction of the persistence mechanism used, providing some specific operations (such as creation, retrieval and removal of instances, among others) without exposing details about connections to the repository. Thus, this pattern can isolate the persistence layer of the business layer. In the next section, this pattern will be illustrated.

3.2. KAO Pattern

The module has an abstract class called *AbstractKAO* responsible for all the abstraction of the KAO pattern. It provides operations for manipulating instances (create, remove, and retrieve) and performing queries. The query methods of the abstract class are protected, that is, only a class that extends AbstractKAO can use these methods. This is intentional, so that the KAO pattern works properly.

Exemplifying the KAO pattern from the user's perspective (Figure 4), who wants to manipulate instances in a given ontology A. This user must create a concrete class (called *OntologyAKAO*) that extends *AsbtractKAO*. This class has concrete methods of creation, retrieval and removal of instances inherits from the abstract class. If the user wants to make a query, he performs the query inside the class *OntologyAKAO*, isolating thereby the persistence layer of the other application layers.

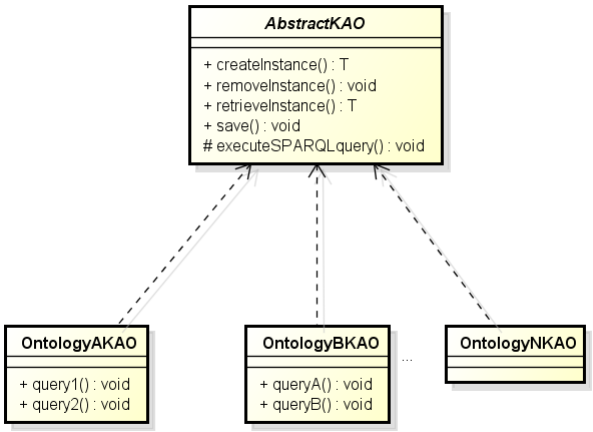

Figure 4. KAO Persistence Pattern

4. Experiment

This section proposes an experiment which analyzes the proposed tool, provides a quantitative and qualitative evaluation. Moreover, the experiment does a comparison between our tool and a tool available in the literature. This comparison aims to verify if the obtained results satisfy the initial proposal, in order words, if the proposed tool increases the efficiency of the programmer development. The experiment was based on [Wohlin et al. 2000] and it has four steps: definition, planning, running and analysis.

4.1. Definition

The first step is determinate the problem which will be analyzed. For this work, the goal is to evaluate the differences on the programmer development. This work does a comparative analysis between the cost to implement a semantic application using the proposed tool and the cost to implement the same application using another tool named Jastor. The experiment occurred at university context and it was done in January 2012. The experiment had eight participants and it focused on the comparison between the proposed tool and the Jastor tool.

4.2. Planning

After the definition of scenario, the next step is planning. Thus, the planning was divided into three main activities: i) creation of control groups; ii) specification of application which will be developed; and iii) specification of the evaluation metrics. The experiment also evaluates the proposed tool and the Jastor tool together with Jena. The choice of these tools is due to two reasons. The first one is that Jena is one of the most popular APIs for manipulating ontologies in Java. The second reason is the tool Jastor is the main tool which works with ontologies in object level with Jena. Therefore, there was this need to integrate the two tools (Jastor and Jena) for this experiment.

The first activity of planning step is the creation of control groups. This experiment had eight undergraduate students of Computer Science and Computer Engineering courses. Several lessons were taught in order to equate the knowledge of participants, thus increasing the experiment control. The lessons were about: i) object-oriented programming and Java language; ii) ontologies modeling using the Protégé environment; iii) queries on ontologies using SPARQL language; and iv) inference on ontologies using SWRL. None of the participants built any ontology-based application before the experiment.

The experiment environment was divided into four control groups, where each group had a pair of students. The first group was named **F5** and its pair used machines with Intel Core i5 CPU processor and 4GB of RAM. The second group was named **F3** and its pair used machines with an Intel Core i3 CPU processor and 4GB of RAM. The **F5** and **F3** groups used the proposed tool. On the other hand, **J5** and **J3** groups used the tools Jastor/Jena and their machines were the same of **F5** and **F3**, respectively. All machines in the experiment performed the operating system Windows 7 Professional 64-bits. The Table 1 summarizes the experiment environment.

After the groups creation, the second activity is the specification of the application which will be developed by the participants. During the experiment, each group will develop the same ontology-based application using its allocated tool. This application

Group	Machine	Allocated Tool
F5	Intel Core i5 CPU processor and 4GB of RAM	Proposed Tool
F3	Intel Core i3 CPU processor and 4GB of RAM	Proposed Tool
J5	Intel Core i5 CPU processor and 4GB of RAM	Jastor/Jena Tool
J3	Intel Core i3 CPU processor and 4GB of RAM	Jastor/Jena Tool

Table 1. The Experiment Environment

uses an adaptation of the *Family.swrl.owl*[3] ontology. This ontology is included in Protégé ontology libraries and it aims to demonstrate the use of SWRL rules in ontology about family relationships. The same Java project will be given to all groups and it contains the main method (*Main*) with features not implemented. The groups goal is run the *Main* method. Therefore, the groups must implement the necessary infrastructure using the allocated tool. The *Main* method has common steps in the development of ontology-based applications, such as: i) add the ontology in repository/database; ii)manipulate instances of ontology; iii)run the SWRL rules; and iv) do SPARQL queries.

The last activity of the planning of the experiment is the specification of the evaluation metrics. As aforementioned, the experiment focuses on the developing efficiency of the participants. Therefore, some quantitative metrics were defined, such as: i) development time, measured in hours; ii) number of lines of implemented code, in this case were not counted: the *Main* lines, the commented lines and the blank lines; iii) performance, measured in milliseconds, iv) memory usage, measured in *Megabytes*; and v) number of errors encountered during development, these errors are Java exceptions that were thrown when a method of the tools was run incorrectly.

In order to have a subjective evaluation about the proposed tool, a questionnaire was elaborated which has questions about the experience of each developer after the end of the experiment. The quantitative and subjective results will be used in the comparison between the tools. The questions are:

1. What did you think about the tool documentation?
2. What did you think about the tool setup?
3. What was the complexity level in the addition of ontology on repository?
4. What was the complexity level in the manipulation of instances?
5. What was the complexity level in the running of the SWRL rules?
6. What was the complexity level in the running SPARQL queries?
7. What did you think that overall the tool?

4.3. Running

The experiment started on January 23th, 2012 and it finished on January 26th, 2012. On the first morning, the experiment was introduced to the participants. At this moment, a general explanation was presented about the experiment and this moment was the first contact between experiment and participants. After the explanation, the pairs were formed and the tools were allocated to each pair (see Table 4.2). Furthermore, the links[4] were

[3]Available at: `http://protege.cim3.net/file/pub/ontologies/family.swrl.owl/family.swrl.owl`

[4]Jastor and Jena documentation: http://jastor.sourceforge.net/ and http://jena.sourceforge.net/documentation.html. Proposed tool documentation: http://jointnees.sourceforge.net/tutorial.html

provided which present the documentation of the tools.

The first data were collected during the experiment. Each pair reported the development time and number of errors encountered. When some team can run the Main method without errors, their code is evaluated. After evaluation, the number of implemented code lines was collected. This count was done manually. On the other hand, the JConsole[5] tool was used to collect the data related to performance and memory usage.

4.4. Analysis of Results

For providing more valuable information about the two target tools, the Table 2 presents the data related to quantitative metrics.

Group	Development Time	Code lines	Performance	Memory Usage	Number of Erros
F5	7 h	72 lines	2584 ms	15,4 MB	3 errors
F3	6 h	81 lines	3757 ms	16,2 MB	1 error
J5	15 h	89 lines	4070 ms	61,5 MB	11 errors
J3	18 h	84 lines	4144 ms	52,2 MB	5 errors

Table 2. The Quantitative Data of the Experiment

Regarding the development time, the proposed tool showed a lower time than the tool Jastor/Jena. On the one hand, the F3 pair finished the experiment in 6 hours and the F5 pair in 7 hours. On the other hand, the J3 pair finished in 18 hours and J5 in 15 hours. The development time is one of the most important factors to evaluate the efficiency on development of ontology-based application. Thus, the proposed tool helps the developers in this issue.

The third column (Table 2) presents the number of code lines for each team. Although, the difference between the pairs was very close when writing code, the proposed tool presented to be considerably more optimized in this issue. This is due to the high level of abstraction which the proposed tool provides.

Then, two factors (performance and memory) are analyzed. Although, these factors are not directly related to efficiency in development, they are very important in the performance of the built application. The performance and the memory usage are measured when the *Main* is running. The performance of the proposed tool was better when we comparing the teams F3 and J5. In other words, the proposed tool had shown the higher performance using a lower machine. The best performance was the F5 team (2.5 seconds) and the worst was J3 team (4.1 seconds). Related to the memory usage, the proposed tool was better because the pairs who used it obtained a significantly smaller cost than the other two pairs. The number of errors metric can be considered the best issue of this work, because the groups who used the proposed tool found four times less number of errors than the others, who used Jastor/Jena.

The Table 3 presents the results of qualitative evaluation. The participants gave a standard note on each question where 1 is the lowest and 5 is the highest. Depending on the question, 1 means terrible and 5 means great. The results show that the proposed tool surpasses the Jastor/Jena tools and they are a reflection of the quantitative results.

[5]Available at: `http://openjdk.java.net/tools/svc/jconsole/`

	Tool	Note 1	Note 2	Note 3	Note 4	Note 5
Question 1	Proposed Tool	0	0	0	4	0
	Jastor/Jena	0	1	2	1	0
Question 2	Proposed Tool	0	0	0	2	2
	Jastor/Jena	0	1	2	1	0
Question 3	Proposed Tool	0	0	0	0	4
	Jastor/Jena	1	2	1	0	0
Question 4	Proposed Tool	0	0	0	3	1
	Jastor/Jena	0	0	3	1	0
Question 5	Proposed Tool	0	0	4	0	0
	Jastor/Jena	1	2	0	1	0
Question 6	Proposed Tool	0	0	0	2	2
	Jastor/Jena	0	0	0	2	2
Question 7	Proposed Tool	0	0	0	2	2
	Jastor/Jena	0	1	1	2	0

Table 3. The Quantitative Data of the Experiment

5. Conclusions and Future Works

This paper aimed to propose a tool to unify several features (manipulate instances, operations on ontologies and repositories, SPARQL query support and inference over SWRL rules) necessary for the development of applications based on ontologies. The tool was presented through its architecture and services description.

As a result, an experiment was conducted to evaluate the tool, focusing on the efficiency of the programmer development. The experiment compared the proposed tool with the tools Jastor and Jena, and both the objective analysis (development time, lines of code, performance, memory and number of errors) and the subjective analysis (through forms) showed better results for the proposed work. Furthermore, although not yet presented in the literature, versions of the proposed tool were used in some works already published, such as [da Silva et al. 2011], [Ferreira et al. 2011], [Holanda et al. 2012] and [Ataide et al. 2011].

However, there are some issues in this work that still need to be addressed. The conducted experiment should be extended to other tools, such OWL2Java and Protege-OWLAPI and with a greater number of developers. The SWRL algorithm must be improved aiming a better performance and further features can be added to the presented tool.

References

Ataide, W., Brito, P., Pedro, A., Costa, E., and Bittencourt, I. I. (2011). A semantic tool to assist authors in the instantiation of software product lines for intelligent tutoring systems context. In *Proceedings of the 4th Workshop on Semantic Web and Education*. WSWEd.

Bard, J. B. L. and Rhee, S. Y. (2004). Ontologies in biology: design, applications and future challenges. *Nature Reviews Genetics*, 5(3):213–222.

Bittencourt, I., Bezerra, C., Nunes, C., Costa, E., Tadeu, M., Nunes, R., Costa, M., and Silva, A. (2006). Ontologia para construção de ambientes interativos de aprendizagem. *Anais do Simpósio Brasileiro de Informática na Educação*, 1(1):567–576.

Bittencourt, I. and Costa, E. (2011). Modelos e ferramentas para a construção de sistemas educacionais adaptativos e semânticos. *Revista Brasileira de Informática na Educação*, 19(01):85.

Bittencourt, I. I., Costa, E., Silva, M., and Soares, E. (2009). A computational model for developing semantic web-based educational systems. *Knowledge-Based Systems*, 22(4):302 – 315.

Buschmann, F., Henney, K., and Schmidt, D. (2007). *Pattern-oriented software architecture: On patterns and pattern languages*. Wiley series in software design patterns. Wiley.

Cheyer, A. and Gruber, T. (2010). Siri: A virtual personal assistant for iphone, an ontology-driven application for the masses. Presentation at Open, International, Virtual Community of Practice on Ontology, Ontological Engineering and Semantic Technology.

da Silva, A., Bittencourt, I., Ataide, W., Holanda, O., Costa, E., Tenório, T., and Brito, P. (2011). An ontology-based model for driving the building of software product lines in an its context. In Dicheva, D., Markov, Z., and Stefanova, E., editors, *Third International Conference on Software, Services and Semantic Technologies S3T 2011*, volume 101 of *Advances in Intelligent and Soft Computing*, pages 155–159. Springer Berlin / Heidelberg.

Das, A., Wu, W., and McGuinness, D. L. (2002). *The emerging semantic web: selected papers from the first Semantic Web Working Symposium*, volume 75 of *Frontiers in artificial intelligence and applications*, chapter Industrial Strength Ontology Management, pages 101 – 118. IOS press.

Ferreira, R., Holanda, O., Melo, J., Bittencourt, I. I., Freitas, F., and Costa, E. (2011). An agent-based semantic web blog crawler. In *Proceedings of the 7th International Conference on Information Technology and Applications*. ICITA, Sydney.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (2004). *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.

Hepp, M., Leenheer, P., Moor, A., and Sure, Y. (2007). *Ontology management: semantic web, semantic web services, and business applications*. Semantic web and beyond. Springer.

Holanda, O., Ferreira, R., Costa, E., Bittencourt, I. I., Melo, J., Peixoto, M., and Tiengo, W. (2012). Educational resources recommendation system based on agents and semantic web for helping students in a virtual learning environment (to appear). *International Journal of Web Based Communities*.

Kiryakov, A., Bishop, B., Ognyanoff, D., Peikov, I., Tashev, Z., and Velkov, R. (2010). The features of bigowlim that enabled the bbc's world cup website. In Krummenacher, R., Aberer, K., and Kiryakov, A., editors, *In proceendings Workshop of Semantic Data Management (SemData)*.

Kiryakov, A., Ognyanov, D., and Manov, D. (2005). Owlim - a pragmatic semantic repository for owl. In Dean, M., Guo, Y., Jun, W., Kaschek, R., Krishnaswamy, S., Pan, Z., and Sheng, Q., editors, *Web Information Systems Engineering - WISE 2005 Workshops*,

volume 3807 of *Lecture Notes in Computer Science*, pages 182–192. Springer Berlin / Heidelberg.

Mika, P. (2007). *Social networks and the Semantic Web*. Semantic web and beyond. Springer.

Szekely, B. and Betz, J. (2009). Jastor: Typesafe, ontology driven rdf access from java. Available from http://jastor.sourceforge.net.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA.