A Survey of Tools for Mapping and Synchronization of Knowledge from Legacy Systems

Helio H. L. C. Monte-Alto, Lucas O. Teixeira and Elisa H. M. Huzita
Informatics Department, State University of Maringá, Av. Colombo 5790, Maringá, Paraná, Brazil

Keywords: OWL Mapping, Knowledge Management, Semantic Web Programming.

Abstract: Knowledge modeling and manipulation are great challenges in the current knowledge-based systems development scenario. Recent development in the area of Semantic Web have arisen with solutions to build intelligent information systems. However, adapting legacy systems to Semantic Web technologies and standards is not trivial and demands too much effort from developers. This paper presents a survey to find tools to ease creation, maintenance and persistence of knowledge by means of mapping between existing application or domain models and OWL ontology concepts. Such tools are intended to be used in a context-aware Global Software Development (GSD) environment using Semantic Web standards to represent context information.

1 INTRODUCTION

The increasing amount of available information and the need for effective communication between heterogeneous information systems have led many applications to adopt formal semantics. Moreover, many software engineers have been trying to improve the interaction of the application with the user, as well as ease knowledge-based systems development. This is possible by taking advantage of the inference capabilities using semantic representation of the application data.

One of the main issues concerning the adoption of formal semantics in legacy applications is the difficulty to adapt the data to this new approach. There are many companies which use legacy systems that are important for them. Requiring such companies to acquire new information systems comprising the needs for knowledge management is not desirable. It would incur in too much additional cost of implementing a new system, implanting it, and adapting to it.

One point also to observe is that nowadays the most used approach for data storage is still relational - or object-relational - databases. However, such technologies do not support very well many capabilities that are required by the current software development scenario. Such requirements include: interoperability, reasonability and availability (Berners-Lee et al., 2001). Several technologies have been proposed to address most of these requirements. They are supported by W3C (World Wide Web Consortium) and adopt an approach often known as the Semantic Web (Berners-Lee et al., 2001).

The main Semantic Web standards for semantic representation are the following: RDF (*Resource Description Framework*), RDFS (or *RDF Schema*) and OWL (*Web Ontology Language*). RDF is a foundation for processing metadata, providing interoperability between applications that exchange machine-understandable information on the Web (Swick and Lassila, 1999). RDFS is an ontology language that provides a simple vocabulary for knowledge representation using RDF (Guha and Brickley, 2004). OWL is an extension of RDFS, which includes a more complex vocabulary for knowledge representation with formal semantics (W3C OWL Working Group, 2009). The OWL vocabulary is based on Description Logics (DL), which allows knowledge representation by means of descriptions of the domain terminology (*Terminology Box*, or TBox) and the information asserted based on the terminology (*Assertion Box*, or ABox).

Although Semantic Web technologies offer several features - mainly interoperability and reasonability - there is still the issue of adapting legacy applications to them. There are many frameworks, such as Jena (Carroll et al., 2004), Sesame (Broekstra et al., 2002) and OWL API (Bechhofer et al., 2003), with focus on manipulating RDF and OWL data. However, there are few mature tools that offer capabilities to create and manipulate such knowledge representation based on existing application and domain

H. L. C. Monte-Alto H., O. Teixeira L. and H. M. Huzita E..
A Survey of Tools for Mapping and Synchronization of Knowledge from Legacy Systems.
DOI: 10.5220/0004436001570164
In *Proceedings of the 15th International Conference on Enterprise Information Systems* (ICEIS-2013), pages 157-164
ISBN: 978-989-8565-60-0

models, such as POJOs, relational databases, ER diagrams, UML diagrams, etc.

Our main goal is to find a solution to map an existing application or domain model to knowledge representation in OWL notation, i.e. a knowledge base (KB) specified by an OWL ontology. Furthermore, we focus on tools that support automatic persistence of entities in a KB, i.e. ABox synchronization and TBox creation and synchronization in a KB. This paper aims to explore some of these tools and find out if there are any of them that meet the most important requirements of our application scenario which is a context-aware Global / Distributed Software Development (GSD/DSD) environment.

This paper is structured as follows. Section 2 presents our application scenario and its issues concerning to knowledge management. Section 3 introduces some of the tools and approaches we have found. Section 4 presents some qualitative comparisons among the most suitable approaches for our application. Section 5 presents a discussion about the suitability of the analysed tools in the application scenario, as well as our choice and some ideas to improve it to better support OWL mapping. Finally, we present the conclusions and future works.

2 APPLICATION SCENARIO: DiSEN

The main goal of our research group is to develop a software engineering environment (SEE) called DiSEN (*Distributed Software Engineering Environment*). It focuses on supporting GSD/DSD, offering features to support communication, persistence and collaboration among geographically distributed teams (Huzita et al., 2007).

Context-awareness and knowledge management are appropriate approaches to support such environment. The first one intends to improve communication and collaboration between individuals. It is necessary that individuals who participate in a project are aware of context information while interacting (Chaves et al., 2008). Context is any information that can be used to characterize the situation of entities that are considered relevant to the interaction between a user and an application, including the user and the application themselves (Dey et al., 2001).

One of the main concerns about context-awareness is the context representation itself. In (Chaves et al., 2011) it is proposed an OWL ontology called OntoDiSEN to specify the context information. Such ontology also specifies a KB for DiSEN since it holds semantic information about the domain. A key requirement for realizing context-aware systems is to give computer systems the ability to understand their situational conditions. To achieve this, it requires that contextual information have been adequately represented for reasoning and machine processing. Ontologies allow to make inferences about the context, since it allows explicit semantic representation (Chen et al., 2004). Therefore, the main characteristic of semantic information on which we are interested in DiSEN is the reasonability.

It has also been proposed in (Monte-Alto et al., 2012) a multi-agent mechanism to implement context awareness, processing and dissemination called ContextP-GSD (*Context Processing on Global Software Development*). One of the main issues of the examples implemented with this mechanism is the difficulty to extract the various context information mainly based on CRUD (Create, Read, Update and Delete) operations. For example, when using the environment to allocate participants to projects, it is necessary to take the following steps to capture the context information: (i) detect the event generated by the user; (ii) traverse the just created entities (currently implemented as JavaBeans) mapping the information to ontology assertions, according to OntoDiSEN's TBox; (iii) send the assertions to another agent to store the context information in DiSEN's KB.

In this case, the problem is the elevated programming and maintainability effort in (ii). Traversing the objects while mapping their contents into ontology assertions requires several lines of code and too much burden on the programmer. Furthermore it is necessary to make it for every operation in the environment.

In the current implementation the entities are mapped to a relational model using JPA (Java Persistence API)[1] through Java Annotations. Based on that, we realized that we could possibly use a similar approach to persist the entities in the ontology. It would allow the environment to automatically update the KB, ensuring its consistency with the application data. Moreover, it would ensure the ontology completeness related to the application, because it would grow together with the application model, becoming more detailed as the mapping becomes more detailed.

Using this new approach, step (ii) would be automatized in cases where context information is related to operations that make changes over application entities. There are context that do not involve such entities, which requires specialized agents to map them to OWL notation. For example, capturing and processing context information about project artifacts - which are managed by external tools like version con-

[1] http://java.sun.com/developer/technicalArticles/J2EE/jpa/

trol systems - or capturing ubiquitous context information, like the presence of a specific user in its workplace. In these cases, it is necessary specific handling to represent the context as ontologies.

In summary, we are interested in a tool that would allow us to use Java Annotations in order to: (i) automatically persist semantic data (**ABox synchronization**); (ii) automatically create part of the TBox specification based on Java Annotations (**TBox generation**); and (iii) continuously synchronize the application domain model and the ontology TBox (**TBox synchronization**).

3 RELATED WORKS

There are many related works focused on Semantic Web application development and ontology-based systems. Many of them are APIs that provide ways to handle and persist knowledge, but lack of features to map it to application code.

Many RDF APIs are available. Some provide access to RDF stores, such as the Jena API (Carroll et al., 2004), the Sesame API (Broekstra et al., 2002) and RDF2Go[2]. Most of these APIs are generic and triple-based, allowing handling RDF triples, although some of them (e.g. Jena) provide different layers to ease handling OWL ontologies. One exception is the OWL API (Bechhofer et al., 2003), which is not based on RDF graphs.

These APIs provide high-level methods for the most common data access patterns. It can be observed three such high-level methods: to *read* object values, to *write* object values and to *find* resources. These access patterns correspond exactly to the main manipulation patterns in relational and object-oriented data. Most of the current works toward mapping existing models of applications to ontologies apply some design patterns, such as Active Record and Data Mapper (Fowler et al., 2002). Fundamentally, these data access patterns are based on filtering the dataset into a set of relevant objects and then manipulating these objects.

Anyway, the existent solutions may be split into two categories: (i) RDB (relational database) mapping and (ii) application code mapping. Some examples of efforts towards the first category are given below:

- **ER2OWL** (Fahad, 2008) is a framework that transforms extended entity-relationship diagram (ERD) into OWL ontology by using a set of predefined rules.
- **D2R** (Bizer and Cyganiak, 2006) is a system which exports relational databases to RDF. This is achieved by assigning URIs to the entries of the database and automatically generating a schema for them.
- **R2RML** (Das et al., 2011) is an ongoing project of *W3C RDB2RDF Working Group* to define a language for expressing customized mappings from relational databases to RDF datasets.

Although these approaches are interesting, there are some issues that make them rather inappropriate for our application scenario. As we intend to maintain some synchronization between the current system model and the ontology, there is a problem because such approach depends on converting data and schema of the RDB continuously to RDF representation. It may generate some overhead because it will be necessary to persist the data primarily in the RDB and then later map this data to RDF. Moreover, the current approaches, except for ER2OWL, are only capable to map from RDB to RDF, whereas we intend to use OWL.

ER2OWL is an interesting approach, however it is necessary to maintain a mapping from the ERD model of the domain to the RDB model, implying in an additional layer of data synchronization and redundancy. There are some efforts concerning the maintenance of such mapping, such as the framework presented in (An et al., 2008). It uses *Round-Trip Engineering* (RTE), a process for synchronizing models by keeping them consistent, thus changes in the relational model are synchronized with the conceptual model (CM). Similarly, such approach may be used to synchronize the CM (in this case, an ERD) with the ontology model. Although this approach incurs in additional overhead after the persistence operation, the results show that the additional time is insignificant (An et al., 2008).

Some solutions that fit in the second category are given below:

- **Jenabean** (Cowan, 2008) is a framework that binds a Java Object specifically to Jena framework to persist JavaBeans using RDFS. It uses Java Annotations to map the objects and does not require placing any interface or extension in the object-oriented model, representing a little intrusive design.
- **JASB (Java Architecture for Semantic Binding)** (Calero, 2010) is a framework which uses Java Annotations for *semantic enforcing*, which consist of reclassifying Java objects at runtime by means of predefined OWL ontologies or SWRL (*Semantic Web Rule Language*) rules. It also in-

[2]http://rdf2go.ontoware.org

Table 1: Qualitative comparisons among Java2OWL, JAOB and JASB.

Feature / technology	Java2OWL	JAOB	JASB
OWL features support	Very good	Satisfactory	Regular
ABox synchronization	Full	Partial	No
Creates TBox from domain model?	Yes	Yes	Yes
TBox synchronization	Partial	Partial	No
Supported APIs	OWLAPI	OWLAPI	Jena
Semantic repositories support	OWLDB	OWLDB	No
SPARQL support	No	No	D/A
License	GPL	LGPL	GPL

cludes a TBox compiler to map the annotated classes to an ontology schema.

- **JAOB (Java Architecture for OWL Binding)** (Malottki, 2008) is a framework, similar to Jenabean, that creates OWL ontologies from Java classes and objects. It uses OWL API instead of Jena, and it does not require any superclass extension or previously defined ontology models. Although it is not yet fully implemented, it provides a good mapping to OWL.
- **Java2OWL** (Ohlbach, 2012) is another Java software library for synchronizing Java class hierarchies with OWL concept hierarchies. It is also based on OWL API. With a few extra annotations in Java class files, the Java2OWL library can automatically map Java class hierarchies to OWL ontologies. The instances of these Java classes are automatically mapped to OWL individuals and vice versa.

There are other approaches that were disconsidered because of the lack of support to OWL, like RDFBeans[3], TRIOO (Fernández et al., 2010) and Texai KB (Reed, 2007). There is also an architecture, proposed in (Paulheim et al., 2011), which allows arbitrary mapping between Java and RDF/OWL in a non-intrusive way by means of using rules to map each class. It was disconsidered because we were unable to find its prototype implementation for testing.

4 ANALYSIS AND EVALUATION OF THE CURRENT TECHNOLOGIES

In Section 3 we discussed some possible existent solutions. OntoDiSEN is modeled as an OWL ontology, thus some of the tools are not appropriate since they only support mapping to RDF triples.

[3]http://rdfbeans.sourceforge.net/

Therefore we chose three frameworks, which are those that promise to support OWL: Java2OWL, JAOB and JASB. Table 1 presents some qualitative comparisons among these technologies, focusing on their appropriateness to DiSEN's scenario.

For our comparison, we chose the following features: (i) OWL features support; (ii) the ABox synchronization issue as exposed in Section 2; (iii) the functionality of creating TBox based on the existing domain model; (iv) the TBox synchronization; (v) the supported APIs; and (vi) the possibility of semantic repositories support. Those features were chosen because they are basically requirements of our scenario. It was also included the license for copying and modifying. Fortunately, they are all free software and open source.

4.1 OWL Features Support

We analysed the OWL features support by taking into account the most important OWL constructions that are implemented in the outlined technologies. A good OWL support must include most of the following items (W3C OWL Working Group, 2009):

- ***Property Axioms:*** property domain and range, functional and inverse functional properties, irreflexive properties, symmetric and asymmetric properties, transitive properties and subproperties;
- ***Class Axioms:*** subclasses, equivalent classes and disjoint classes;
- ***Property Expressions:*** inverse properties;
- ***Class Expressions:*** Boolean combinations (intersection, union, complement) and existential and universal restrictions;

Supporting the mapping of all these features is a great challenge since there are many semantic differences between the object-oriented approach and the Description Logics implemented in OWL. Most of the class axioms and expressions are possibly the most difficult to map from Java classes, as it is shown in Table 2.

Table 2: Differences between object-oriented approach and the description logics.

Characteristic / approach	Java	Description logics
Class hierarchy	Mono-inheritance	Multi-inheritance
Class instances	Each object is instance of exactly one class	An individual can be instance of several classes
Class equivalence	No	Yes
Property restrictions /anonymous classes	No [1]	Yes
Class union, intersection and complement	No	Yes
Runtime evolution	class definitions typically cannot evolve during runtime	data is integrated from heterogeneous sources with varying structures where both schema and data may evolve at runtime

[1]There are anonymous classes in Java, but not in the same sense as in DL. In DL, anonymous classes are property restrictions that describe a class of all individuals that satisfy the restrictions. In Java, anonymous classes are inner classes that you can declare inside a method without naming them.

Java2OWL is the tool that provides the best efforts to address such problems. To deal with the classes hierarchy and instances it is proposed in (Ohlbach, 2012) the *individual wrappers*, which encapsulate an OWL individual together with several Java objects. The main problem of this approach is that it is too much intrusive for legacy systems, in the sense that Java programs should not work with Java objects in the usual way, but with individual wrappers.

Another approach, not related to Java2OWL, proposed in (Kalyanpur et al., 2004) is to use Java interfaces instead of simple Java classes to be mapped to OWL, since an interface can inherit from many others. This way, each OWL class corresponds to a Java interface and a JavaBean that implements the access methods (get/set methods) declared in such interface. Using this approach it is possible to address the issue of hierarchy, equivalence, union, intersection and complement. Although this is not as intrusive as the individual wrappers, there is the need for creating interfaces for every class, which may be done by using an interface extractor like the one provided in NetBeans IDE [4]. Unfortunately, we have not found any tools that support Java to OWL mapping using this approach.

Although not complete, JAOB also provides a good mapping from Java to OWL. It does not try to deal with the class axioms and expressions as thorough as (Ohlbach, 2012) and (Kalyanpur et al., 2004), but provides concise ways to map Java classes to their representation in OWL.

JASB is much more incomplete than the others. Moreover, its goal is not to map Java to OWL, but reclassify Java objects at runtime - which the authors call *semantic enforcing* - based on predefined ontologies or rules.

[4]http://netbeans.org

4.2 ABox Synchronization

The ABox synchronization means that changes in Java objects' attributes are forwarded to the corresponding OWL individuals. In other words, the application data is continuously and automatically mapped and persisted in the KB represented by the OWL ontology.

The Java2OWL library provides two ways to do it (Ohlbach, 2012):

- **Life Synchronization:** in this approach all changes to the attributes of Java objects are immediately forwarded to the ontology. This is possible if the annotated Java classes got a "synchronizer code" injected. Such code is automatically injected by activating the `J2OSynchronizerAgent` or by setting a specific flag to `true`.
- **Block Synchronization:** in this approach the application data is only synchronized when a specific method is called to commit the changes to the ontology. It does not require any code to be injected in the Java classes.

JAOB does not have a specialized synchronization code. It has only a class called `Marshaller` which is responsible for creating an OWL serialization of the application data. Such serialization includes TBox and ABox. The main problem of this implementation concerns the removal operations, because marshalling consists of appending new data to the previous serialization. Therefore it still has to be improved to support better synchronization between the application data and the KB.

As JASB does not aim to provide semantic persistence for Java applications, it does not have any mechanism to do it.

4.3 TBox Generation and Synchronization

This feature consists of generating the terminology of the ontology based on the Java Annotations. It would allow legacy systems developers to model the ontology and map it to application model using only Java annotations, which is much easier than doing both things by hand separately. TBox synchronization is also a desirable feature as it supports software evolution.

In DiSEN's scenario we already have a fairly advanced model of the domain ontology. However, TBox synchronization would be very convenient since OntoDiSEN is still under development. It is possible to use the TBox synchronization with the Java Annotations to continuously construct the ontology together with the application, thus making it easier to find out semantic details that could be added to the ontology model. Such details could improve the reasoning capabilities for the KB, enabling smarter context processing.

Java2OWL assumes the existence of a *background ontology* with predefined classes and properties. The Java2OWL compiler then extend it with OWL classes generated from Java classes, creating an *extended ontology* (Ohlbach, 2012). However, such *background ontology* is optional, which means that it is possible to create a new TBox from scratch only with the OWL generated from Java classes. The synchronization is partially supported by using the previous *extended ontology* as *background ontology* when it is necessary to make changes in the Java code.

JAOB and its marshalling approach also generate TBox based on the Java Annotations. The synchronization can be partially guaranteed because the marshalling operation creates both TBox and ABox. Although JASB does not aim to synchronize application data with ontologies, it provides a compiler to generate TBox based on the Java classes.

In every instance, it is necessary to recompile the TBox if the class structure is changed (e.g. a class is removed, or a hierarchy relation is changed). Such changes may end up incurring inconsistencies or loss of knowledge, depending on the affected structure. Inconsistency can be resolved by using consistency checking provided by inference engines, as well as techniques like defeasible logic (Bao and Honavar, 2004).

4.4 Supported APIs and Semantic Repositories

For legacy systems without any implementation of semantic features there are not many issues concerning the technologies and APIs which are going to be used. As there are not many available frameworks and toolkits for manipulating OWL and most of them are still under development, the options are very strict. All of the technologies in question, except JASB, use OWL API, which is quite mature, although there are some downsides related to other technologies, like Jena. However, further comparison of such tools is out of the scope of this paper.

Our main concern about the supported APIs is mainly due to the fact that DiSEN already has some implemented features that use ontologies. These are currently implemented with Jena framework. Among the analysed technologies, only JASB supports Jena, but as it is seen in the previous sections, it is not appropriate for our problem.

Another consideration is SPARQL (Prud'hommeaux and Seaborne, 2008) and triple store support. Different from Jena, OWL API bypasses RDF graphs, i.e. it is not possible to deal with lower level RDF triples. Therefore, currently it is difficult to support SPARQL queries and triple stores - which are high performance solutions for storing and accessing semantic data - in OWL API.

An alternative for SPARQL and triple stores is OWLDB (Henss et al., 2009). It consists of an extension to OWL API which provides native mapping of OWL constructs to a database schema without a complex transformation in triples. According to performance tests reported in (Henss et al., 2009) the system performs comparably to some triple-based approaches and even outperforms those on several queries.

5 DISCUSSION

In the previous section some important characteristics of the selected solutions were enumerated, in order to find out what would be the most suitable for our application scenario. Given the good evaluation of Java2OWL, and considering it is a recent project with plenty of documentation, it is reasonable to go deeper into this library. JAOB is also a very interesting initiative, however it is a discontinued project lacking of documentation and publications.

The most important issue about adopting Java2OWL in our scenario is the incompatibility with the API we are currently using for dealing

with ontologies. It is necessary to evaluate what would be the best alternative: creating a new version of Java2OWL which uses Jena or refactoring the current code of ContextP-GSD and related projects to use OWL API. Considering the contribution to the Semantic Web community, we conclude that creating a new version of Java2OWL is the best alternative. It would also allow us to use SPARQL queries and reasoning features and the various triple store solutions.

There are still some complex issues related to OWL, like dealing with multi-inheritance in a less intrusive way, dealing with inconsistencies in the ontology, and considering open and closed world assumptions, that could be better explored as we advance in working with the tool.

6 CONCLUSIONS AND FUTURE WORKS

It was presented a qualitative comparison among some solutions to map application data to OWL representation by means of Java Annotations. This facilitates the incorporation of semantic features in legacy systems in a little intrusive way. It also reduces the programming effort since it eliminates the need to write code responsible to map specific Java classes to OWL concepts for every internal operation which involves manipulating application entities. Another advantage is that such mapping approach allows any developer to cumulatively construct the ontology TBox as the mapping becomes more detailed.

Among many researched approaches, there are two outstanding technologies: Java2OWL and JAOB. The first one showed to be the most appropriate for DiSEN's application scenario, because of its wide OWL features support and its focus on synchronization between application data and OWL. Moreover, Java2OWL is an ongoing project whereas JAOB is not being supported.

Although many advantages were encountered in these approaches, there are still some issues. First, the mapping does not cover all of the OWL vocabulary yet. Indeed such mapping is not trivial. Therefore it still needs a lot of improvements. Second, such approaches only support OWL API, which is not suitable when it is intended to use SPARQL queries and/or triple stores. In DiSEN's case, it becomes even more serious since there are already many implemented semantic features using Jena instead of OWL API.

For our case, it is necessary to make a choice: migrating to OWL API or implementing an extension of Java2OWL or JAOB to support Jena. The first one is interesting, however it would be necessary to discard the increasing progress of technologies based on RDF triples, such as SPARQL and triple stores. Benchmarks like Berlin SPARQL Benchmark (BSBM) (Bizer and Schultz, 2009) are continuously exposing the performance improvement made on triple stores and SPARQL engines. Though there is the OWLDB alternative for persistence and querying, it may not be appropriate to ignore all of the current work being done on triple stores.

Based on that we conclude that the best choice is to implement an extension of Java2OWL which supports Jena. It would allow us to take advantage of triple-based technologies without losing the possibility of using OWL API and OWLDB. Furthermore, we intend to gradually improve such tool in order to adjust it for our needs.

REFERENCES

An, Y., Hu, X., and Song, I.-Y. (2008). Round-trip engineering for maintaining conceptual-relational mappings. In *CAiSE*, volume 5074 of *Lecture Notes in Computer Science*, pages 296–311. Springer.

Bao, J. and Honavar, V. (2004). Collaborative ontology building with wiki@nt - a multi-agent based ontology building environment. In *Third International Workshop on Evaluation of Ontology Building Tools, Hiroshima*.

Bechhofer, S., Volz, R., and Lord, P. W. (2003). Cooking the semantic web with the OWL API. In *The Semantic Web – ISWC 2003: Second International Semantic Web Conference, Sanibel Island, FL, USA*, volume 2870 of *Lecture Notes in Computer Science*, pages 659–675. Springer, Berlin.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43.

Bizer, C. and Cyganiak, R. (2006). Dr2 server - publishing relational databases on the semantic web. Poster at the 5th International Semantic Web Conference (ISWC2006).

Bizer, C. and Schultz, A. (2009). The Berlin SPARQL benchmark. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(2):1–24.

Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *Proceedings of the first Int'l Semantic Web Conference (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68, Sardinia, Italy. Springer Verlag.

Calero, J. M. A. (2010). Jasb project. http://jasb.sourceforge.net/. Accessed May 2012.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K. (2004). Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide*

Web conference on Alternate track papers & posters, WWW Alt. '04, pages 74–83, New York, NY, USA. ACM.

Chaves, A., Wiese, I., da Silva, C., and Huzita, E. (2008). A model based on context-awareness for information dissemination in a distributed software development environment (in portuguese). In *XXXIV Conferencia Latinoamericana de Informáica (CLEI 2008), Santa Fe, Argentina*, pages 1365–374.

Chaves, A. P., Steinmacher, I., Leal, G. C. L., Huzita, E. H. M., and Biasão, A. B. (2011). Ontodisenv1: an ontology to support global software development (in portuguese). *CLEI Electron. J.*, 14(2).

Chen, H., Finin, T. W., and Joshi, A. (2004). Semantic web in the context broker architecture. In *PerCom*, pages 277–286. IEEE Computer Society.

Cowan, T. (2008). Jenabean: Easily bind javabeans to rdf. http://www.ibm.com/developerworks/java/library/j-jenabean.html. Accessed May 2012.

Das, S., Sundara, S., and Cyganiak, R. (2011). R2rml: Rdb to rdf mapping language (w3c working draft).

Dey, A., Abowd, G., and Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16(2):97–166.

Fahad, M. (2008). Er2owl: Generating owl ontology from er diagram. In *Intelligent Information Processing*, volume 288 of *IFIP*, pages 28–37. Springer.

Fernández, S., Berrueta, D., Rodríguez, M. G., and Gayo, J. E. L. (2010). Trioo - Keeping the Semantics of Data Safe and Sound into Object-oriented Software. In *International Conference on Software and Data Technologies*, pages 311–320.

Fowler, M., Rice, D., and Foemmel, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

Guha, R. V. and Brickley, D. (2004). RDF vocabulary description language 1.0: RDF schema. W3C recommendation, W3C.

Henss, J., Kleb, J., Grimm, S., and Bock, J. (2009). A database backend for owl. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2009), Chantilly, VA, United States, October 23-24, 2009*, volume 529.

Huzita, E., Tait, T., Colanzi, T. E., and Quináia, M. (2007). Disen - a distributed software development environment (in portuguese). In *1st Workshop de Desenvolvimento Distribuído de Software*.

Kalyanpur, A., Pastor, D. J., Battle, S., and Padget, J. A. (2004). Automatic mapping of owl ontologies into java. In *SEKE*, pages 98–103.

Malottki, J. (2008). Yoshtec kb: Jaob (java architecture for owl binding). http://wiki.yoshtec.com/jaob. Accessed May 2012.

Monte-Alto, H. H. L. C., Biasão, A. B., Teixeira, L. O., and Huzita, E. H. M. (2012). Multi-agent applications in a context-aware global software development environment. In *Distributed Computing and Artificial Intelligence*, volume 151 of *Advances in Intelligent and Soft Computing*, pages 265–272. Springer Berlin / Heidelberg.

Ohlbach, H. J. (2012). Java2owl - a system for synchronising java and owl version 1.1. Technical report, Institute for Informatics, University of Munich.

Paulheim, H., Plendl, R., Probst, F., and Oberle, D. (2011). Mapping pragmatic class models to reference ontologies. In *ICDE Workshops*, pages 200–205. IEEE.

Prud'hommeaux, E. and Seaborne, A. (2008). SPARQL query language for RDF. W3C recommendation, W3C.

Reed, S. L. (2007). Semantic annotation for persistence. In *Proceedings of AAAI2007 Workshop on Semantic e-Science*.

Swick, R. R. and Lassila, O. (1999). Resource description framework (RDF) model and syntax specification. supersed work, W3C.

W3C OWL Working Group (2009). OWL 2 web ontology language document overview. Technical report, W3C.