Using Ontologies for Software Development Knowledge Reuse

Bruno Antunes, Nuno Seco and Paulo Gomes

Centro de Informatica e Sistemas da Universidade de Coimbra
Departamento de Engenharia Informatica, Universidade de Coimbra
bema@student.dei.uc.pt, {nseco,pgomes}@dei.uc.pt
http://ailab.dei.uc.pt

Abstract. As software systems become bigger and more complex, software developers need to cope with a growing amount of information and knowledge. The knowledge generated during the software development process can be a valuable asset for a software company. But in order to take advantage of this knowledge, the company must store and manage it for reuse. Ontologies are a powerful mechanism for representing knowledge and encoding its meaning. These structures can be used to model and represent the knowledge, stored in a knowledge management system, and classify it according to the knowledge domain that the system supports. This paper describes the Semantic Reuse System (SRS), which takes advantage of ontologies, represented using the knowledge representation languages of the Semantic Web, for software development knowledge reuse. We describe how this knowledge is stored and the reasoning mechanisms that support the reuse.

Key words: Ontologies, Sematic Web, Software Reuse, Knowledge Management.

1 Introduction

The new vision of the web, the Semantic Web [1], aims to turn the web more suitable for machines, thus make it more useful for humans. It brings mechanisms that can be used to classify information and characterize it's context. This is mainly done using knowledge representation languages that create explicitly domain conceptualizations, such as ontologies [2]. These mechanisms enable the development of solutions that facilitate the access and exchange of relevant information.

The storage and access to relevant information is a central issue in knowledge management [3], which comprises a set of operations directed to the management of knowledge within an organization, helping the organization in the achievement of its objectives. The technologies that are behind the Semantic Web vision provide an opportunity to increase the efficiency of knowledge management systems, turning them more valuable. These technologies provide knowledge representation structures that can be used to improve the storage, search and retrieval functionalities of common knowledge management systems.

One of the problems that software development companies face today is the increasing dimension of software systems. Software development projects have grown by complexity and size, as well as in the number of functionalities and technologies that are involved. The resources produced during the design and implementation phases of such projects, are an important source of knowledge inside software development companies. These resources contain the knowledge that has been used to solve various development problems of past projects. Some of these problems will certainly appear in the future, for this reason the software development resources produced in the past, represent an important source of knowledge that can be reused in the future. In order to make this knowledge useful and easily shareable, efficient knowledge management systems are needed. These knowledge management systems must be able to efficiently store, manage, search and retrieve all kinds of knowledge produced in the development of software systems. To accomplish this, the stored knowledge must be well described, classified and accessible from where it is needed.

As referred before, ontologies are one of the most important concepts present in the Semantic Web architecture. They are a powerful mechanism for representing knowledge and encoding its meaning, allowing the exchange of information that machines are able to process and concisely understand. These structures can be used to model and represent the knowledge stored in a knowledge managemente system and classify it according to the knowledge domain that the system supports. In this paper we describe the *Semantic Reuse System* (SRS), a system for the reuse of software development knowledge that we are developing at the Artificial Intelligence Laboratory (AILab) of the University of Coimbra.

The goal of SRS is the management and reuse of software development knowledge using the mechanisms provided by the Semantic Web, such as RDF, RDFS and OWL [4], to represent the ontologies that reflect the knowledge used by the system. We make use of a representation ontology, used to represent tne different types of artifacts that the system deals with, and a domain ontology, to classify this these artifacs. As software development knowledge, we consider the various elements that result from the software development process, such as specification documents, design diagrams, source code, etc. We will name each one of these elements as a *Software Development Knowledge Element* (SDKE). Our approach aims to provide efficient mechanisms for storing, searching, retrieving and managing the stored knowledge, using ontologies and the Semantic Web languages to represent them.

The next section describes the SRS architecture, as well as the main functionalities and requirements of the system. Section 3 describes the knowledge base used in the system, and its subcomponents. In section 4 the knowledge reuse mechanisms are described and illustrated. Section 5 presents similar works to SRS and section 6 concludes the paper with final remarks and some future work.

2 SRS Architecture

The SRS system works as a platform that provides a way to store the software development knowledge in the form of SDKE's. Besides being stored in the file system and accessible through an interface framework, the different elements should be described using a *Representation Ontology* and classified through a *Domain Ontology*. These additional knowledge representation structures will be used to empower the search mechanisms and make easier the reuse of stored knowledge. To take advantage of the system functionalities, a set of client applications can be developed. These client applications can be directed to users and developers, implemented as standalone applications or as plug-in's for well known development environments, or to those responsible for the management of the stored knowledge and supporting structures of the platform. The system provides an API for these applications using a web service, for better portability and integration.

In figure 1, we present the system's architecture. The system can be structured in three different logical layers: data, core and interface. The data layer corresponds to the knowledge base and is described in section 3, it stores all the knowledge needed for the system reasoning, including the knowledge representation structures and the SDKE's. The core layer implements the reasoning mechanisms, including the search, suggestion and browsing facilities, which are described in section 4. The interface layer comprises the semantic web service and the applications that use the system's functionalities through this semantic web service.

The *Management Tools* comprise a set of applications that can be used to manage the system and its knowledge base. There is a *Knowledge Base Manager* that is used to manage the knowledge representation structures, both the *Domain Ontology* and the *Representation Ontology*, the *SDKE Repository* and the system related data, such as logging and accounting. These applications are intended to be used mainly by knowledge managers or experts.

The *User Tools* comprise a set of applications that will use the services provided by the platform. These may include standalone applications specially designed to use the system's framework, or plug-in's for development tools that integrate the system's functionalities into the environment of the application. An add-in for the well known integrated development environment from *Microsoft*, *Visual Studio 2005*, will be developed. Through this add-in we will integrate the functionalities of the platform into the application, enabling searching, submission and reuse of the stored knowledge. Besides these operations, we intend to provide pro-active suggestion of knowledge based on the user interaction with the application. By monitoring the user actions and the underlying working context, the system may suggest knowledge that can be reused in that context.

The *Semantic Web Service* interfaces the *Core Layer* to all the client applications, including both the *Management Tools* and the *User Tools*. The web service provides an API that maps to the *Core API* and interfaces all the functionalities of the system, including the functionalities related to the search, storage and reuse of knowledge, as well as those related to the management of the system.

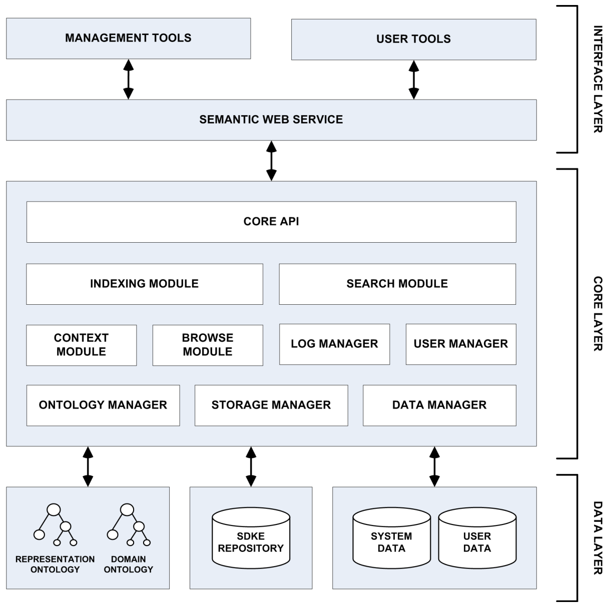


Fig. 1. The architecture of SRS.

The use of a web service as an interface to the platform enables portability and integration. Specially, the Semantic Web Services [5] are an attempt to make web services prepared to be efficiently used by software agents. This is done by providing agent-independent mechanisms to attach metadata to a web service that describes its specification, capabilities, interface, execution, prerequisites and consequences. The OWL-S [6] language has been developed in order to support these mechanisms.

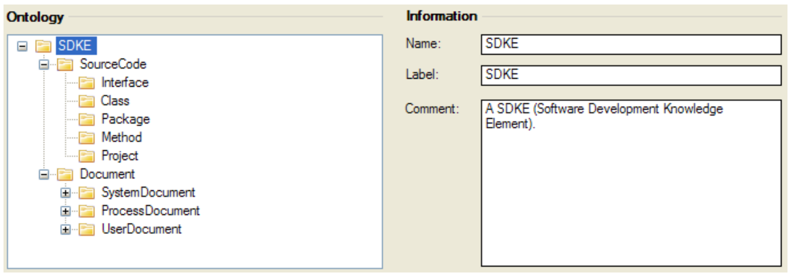
3 Knowledge Base

This section describes the knowledge base used in SRS. It comprises the *Representation Ontology*, the *Domain Ontology*, the *SDKE Repository*, and *System and User's Data*. The ontologies in the platform will be represented using languages from the Semantic Web, namely RDF, RDFS and OWL [4], and will be managed using the *Jena Semantic Framework* [7] (`http://jena.sourceforge.net/`), which is an open source Java framework for building Semantic Web applications, providing an API for manipulating RDF graphs, including support for RDFS and OWL. The *SDKE Repository* will be managed using *Apache Lucene* [8](`http://lucene.apache.org/`), which provides an open source high-performance engine for full-featured text search and document storage written in Java. The system and user data will be managed through a relational database. The next subsections describe each of these parts.

3.1 Representation Ontology

The *Representation Ontology* defines a set of classes, properties and relations between classes that are used to model and describe the different types of SDKE's. This ontology is used to define every type of SDKE's that can be used in the system. The different properties that apply to each one of them and the specific relations that can exist between them.

This ontology is specified taking into account the different types of elements that we want to store and reuse. This is mainly done before the deployment of the system. But this ontology should be viewed as a dynamic structure, since it should be possible to do some adjustments to the model after it has been deployed, without affecting the system as a whole. The *Representation Ontology* will also include the metadata model developed by the *Dublin Core Metadata Initiative* (DCMI) [9], which is an organization engaged in the development and promotion of interoperable metadata standards for describing resources that enable more intelligent information systems. The DCMI is integrated in the representation ontology as annotation properties to SDKE objects. The adoption of these standards enables the system to easily exchange data with other platforms, or even with intelligent agents. As important as describing the components that are being stored for reusing, is the association, to these components, of what we call of reuse documentation. This reuse documentation will be defined in the *Representation Ontology*, using a metadata model based on the work of Sametinger [10], providing vital information for those who will reuse the stored knowledge in future development projects. In figure 2, we present an example of part of the *Representation Ontology*. Note that what appears in the figure is the ontology editor of the *Knowledge Base Manager*, which is one of the management tools of SRS. The figure represents part of the representation ontology taxonomy.



Name: SDKE
Label: SDKE
Comment: A SDKE (Software Development Knowledge Element).

Fig. 2. Part of the representation ontology used in SRS.

3.2 Domain Ontology

The *Domain Ontology* is used to represent the knowledge of the domain that the system supports. It is comprised of concepts and relations between concepts.

The SDKE's stored in the platform are indexed to one or more concepts in this ontology, reflecting the knowledge stored in each one of them. Having relations that semantically relate concepts with other concepts, the system is able to retrieve not only knowledge that match a query, but also knowledge that is semantically related and eventually relevant to the user.

This ontology is dynamically built during the submission of new knowledge to the platform. To construct this ontology, the system uses other source ontologies, from which it retrieves the concepts and relations that are needed to index new knowledge. One of the source ontologies that is used is *WordNet* [11], which is a large lexical database of English. In *WordNet*, nouns, verbs, adjectives and adverbs are grouped into sets of synonyms, each expressing a distinct concept. The system retrieves the concepts from WordNet, that it needs too classify the knowledge supported, and import them to the domain ontology. But the system may also search, retrieve and import ontologies that contain the desired concepts, using tools such as the *Swoogle Semantic Web Search Engine* [12], which is a crawler-based indexing and retrieval system for documents containing knowledge representation structures of the Semantic Web. In figure 3, we present an example of part of the *Domain Ontology*. As in the previous figure, what appears in the figure is the ontology editor of the *Knowledge Base Manager*, which is one of the management tools of SRS. The figure represents part of the domain ontology taxonomy.

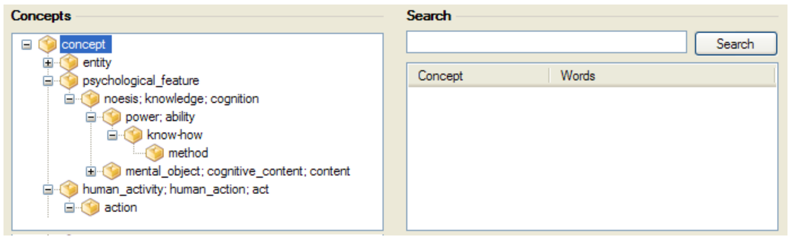

Concepts
concept
entity
psychological_feature
noesis; knowledge; cognition
power; ability
know-how
method
mental_object; cognitive_content; content
human_activity; human_action; act
action
Search
Search
Concept
Words

Fig. 3. Part of the domain ontology used in SRS.

In order to import new concepts, from source ontologies to the *Domain Ontology*, the system uses techniques of ontology integration, which is a current field of research and where many problems are still unsolved. As described in the work of Pinto, et al, [13] ontology integration can be used in three different situations: integration of ontologies when building a new ontology reusing other available ontologies; integration of ontologies by merging different ontologies about the same subject into a single one that "unifies" all of them; and the integration of ontologies into applications. From these, the situation we face in our system is building a new ontology reusing other available ontologies. Specially, we want to build our *Domain Ontology* by importing the concepts and their relations, from source ontologies, as long as we need them to index new knowledge. By using *WordNet* as the main knowledge source, our *Domain Ontology* could be seen as a small part of *WordNet*, since we would only bring to our ontology the concepts needed to reflect the stored knowledge. But, having *WordNet* as the only source ontology creates some problems, such as domain vagueness, since it is general, and language specificness, since it is in English. Another alternative would be reusing ontologies available in large repositories or in the Internet. This approach, although partially solving the problem of domain vagueness and language specific, brings even more complex challenges, such as ontology evaluation, ontology ranking, ontology segmentation and ontology merging, which have been described by Alani [14].

3.3 SDKE Repository

The *SDKE Repository* represents the knowledge repository in the platform. When submitted, the development knowledge is described and indexed, using the knowledge representation structures described before, and is stored in the repository in the same form it as been submitted. This way, the repository contains all the files that have been submitted to the platform, representing the SDKE's indexed by the system. These files are stored using *Apache Lucene*, as referred before, which also provides advanced text search capabilities over the stored documents.

3.4 System and User's Data

The *System and User's Data* is comprised of data related to system configuration, logging and accounting. This data comprises parameters and other configuration information used by the system; logging data generated by the user's interaction with the system, which can be used to analyze the user's behavior and eventually improve the system's efficiency and precision; and data for user accounting, which includes user accounts, user groups, access permissions, etc.

4 Knowledge Reuse

In the previous section we have described the knowledge that the system uses and stores, this section presents the reasoning mechanisms used for manipulating this knowledge. There are a set of modules that implement the basic operations on the platform: the *Core API* is an API to all the core functionalities, and acts as an interface between the core modules and the *Semantic Web Service*; the *Log Manager* implements an interface to all the operations related to system's logging; the *User Manager* implements an interface to all the operations related to users; the *Ontology Manager* implements an interface to all the operations over the *Representation Ontology* and the *Domain Ontology*; the *Storage Manager* implements an interface to all the operations over the *SDKE Repository*; and the *Data Manager* implements an interface to all the operations over the *System Data* and *User Data*.

In the next sub-sections, we describe in detail the most relevant modules present in the *Core Layer*: *Indexing Module*, *Context Module*, *Search Module* and *Browse Module*. These modules implement the operations related to the submission, search, retrieval and management of knowledge in the platform.

4.1 Indexing Module

The *Indexing Module* implements and provides an interface to submit and index new knowledge to the platform. The submitted knowledge, in the form of software development artifacts, must be described using the *Representation Ontology*, indexed to concepts in the *Domain Ontology* and stored in the *SDKE Repository*.

The description of the artifacts using the *Representation Ontology* is done taking in account their type and associated metadata. The *Representation Ontology* defines the different types of artifacts that can be stored in the platform, their relations and associated metadata. Given the artifacts and the corresponding metadata, the system creates instances of classes in the *Representation Ontology* that represents each one of the artifacts. The indexing of the artifacts in the *Domain Ontology* comprises two phases: first the system must extract from the artifacts the concepts used for indexing and then it must verify if those concepts exist in the domain ontology or if they must be imported from source ontologies. The extraction of concepts from the artifacts is done using linguistic tools from *Natural Language Processing* (NLP) [15]. To index the SDKE in the *Domain Ontology*, the system first searches the *WordNet* [11] for synsets containing the base forms of the relevant terms of the SDKE. The synsets that contain the relevant terms are then imported to the *Domain Ontology*. The synsets are imported in conjunction with the whole set of its hypernyms, which form one or more paths from the synset to the root of the synset's hierarchy. The hypernym of a concept is a concept that is more general that the first and that contains its definition. The terms not found in *WordNet* are added to the root of the concepts hierarchy in the *Domain Ontology*. These concepts are added with a special property indicating that they were not imported from the *WordNet* and are waiting the validation of the knowledge manager. With all the concepts that classify the SDKE added to the *Domain Ontology*, the system indexes the SDKE to these concepts. After being described and indexed, the artifacts must be stored in the *SDKE Repository*.

4.2 Context Module

The *Context Module* implements and provides an interface to retrieve knowledge that is relevant for a development context. As development context we consider a set of elements that can be found in a common IDE, such as packages, interfaces, classes, methods, etc, and an event, such as the creation of any of the referred elements. The process of interaction between the user and the *Context Module* is represented in figure 4.

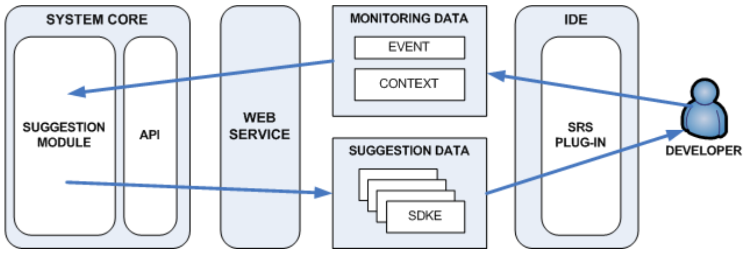


Fig. 4. The process of interaction between the user and the context module.

The *Context Module* can be useful to suggest knowledge for reuse in a development environment, such as a programming IDE. By using this module, an IDE can pro-actively suggest knowledge for reuse upon certain actions of the developer. For instance, when the developer creates a new package, the IDE can use the *Context Module* to retrieve knowledge that is relevant to the package being created taking into account the context that surrounds that package. As relevant knowledge, we can consider two kinds of knowledge: suggested knowledge and related knowledge. The suggested knowledge represents knowledge suitable for replacing the element that fired up the event, so if the developer is creating a package the system will suggest packages that can be reused. The related knowledge represents knowledge that cannot replace the element that fired up the event, but is somehow related to that element or its context, and can also be useful to the developer. For instance, if a user is modifying a class, the system can suggest the piece of text from the specification documentat where the class is described. The retrieval of SDKE's, through keyword-based search or context-based search, is implemented by the *Search Module*, which we describe next.

4.3 Search Module

The *Search Module* implements and provides an interface to all the operations of searching and retrieving stored knowledge. Three different search methods were developed: keyword-based search, ontology-based search and full search. The process of interaction between the user and the *Search Module* is represented in figure 5.

The keyword-based search relies on the full-text search capabilities of the *Apache Lucene* [8] engine that supports the *SDKE Repository.* When SDKE's are stored in the repository, the *Apache Lucene* engine indexes their textual data so that keyword-based searches can be made in the repository. The results returned are ranked according to the internal algorithms of the *Apache Lucene* engine, reflecting their relevance to the query applied.

The ontology-based search uses the *Domain Ontology* to retrieve the SDKEs that are relevant to a query. To accomplish this, the system splits the query into

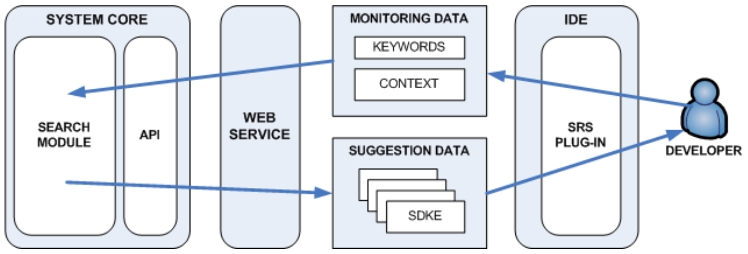


Fig. 5. The process of interaction between the user and the search module.

terms and uses some rules of detachment, the same used in *WordNet* [11], to get the base form of each term. It then searches for concepts that are referenced by the base forms of these terms. The system retrieves all the SDKE's indexed by the concepts that are referenced by the terms of the search query. If the number of SDKE's retrieved is not enough, the concepts are expanded to their hyponyms and hypernyms. Then the SDKE's indexed by the expanded concepts are retrieved and again, if they are not enough, the process is repeated until a defined maximum expansion length is reached. Finally, the retrieved SDKE's are ranked according to the number of concepts they are indexed to.

4.4 Browse Module

The *Browse Module* provides an interface to all the operations related to the browsing of the stored knowledge. This can be done using either the *Representation Ontology* or the *Domain Ontology*. This means that the knowledge stored in the platform can be organized by its type or by the concepts it is associated with.

5 Related Work

This section introduces some recent works in the field of knowledge management for software development using Semantic Web technologies.

In their work, Hyland-Wood, et al, [16] describe ongoing research to develop a methodology for software maintenance using Semantic Web techniques. They propose a collection of metadata for software systems that include functional and non-functional requirements documentation, metrics, success or failure of tests and the means by which various components interact. By recording and tracking the changes in the metadata it is possible to notify the developers of changes that may influence further development. The system metadata is encoded using Semantic Web techniques such as RDF, OWL and SPARQL Query Language [17].

In order to separate the software engineering domain knowledge from the operational knowledge, Hyland-Wood [18] have created an OWL ontology of software engineering concepts (SEC Ontology). This ontology describes relationships between object-oriented software components, software tests, metrics, requirements and their relationships to the various software components. They have shown the validity of their approach through an example implementation using data representing a small portion of a real-world software package. They applied SPARQL queries to this example data to show that properties representing the last modification of components and the last validation of requirements could be updated and that subsequent queries could be used to determine state changes. Based on this grounding work, they envision a software maintenance methodology where developers could manually input information about requirements through some kind of IDE which would store software system metadata in a RDF graph and accept SPARQL queries. Changes in metadata could be tracked and developers would be notified when these changes required actions.

Recent work by Thaddeus [19] has proposed a system similar to SRS, which uses formal logics for representing software engineering knowledge and reasoning. The proposed system, uses a multi-agent approach to implement several reasoning mechanisms, like: knowledge access, process workflow automation or automated code generation. They use three different ontologies: Project Ontology, Process Ontology and Product Ontology. This system has some common goals with SRS, but has a different approach to the overall architecture. While Thaddeus uses a multi-agent approach, SRS uses a Producer/Consumer approach based on several types of clients that can be connect to the main system through a Semantic Web Service.

6 Conclusions

In this paper we described the SRS system, which intends to explore new ways to store and reuse knowledge. We have used ontologies and the Semantic Web technologies to build a software development reuse platform, representing and storing knowledge in RDF, RDFS and OWL. The SRS platform provides several ways to reuse and share knowledge, including a pro-active way of suggesting relevant knowledge to the software developer using the user context.

Ontologies and the Semantic Web technologies enable the association of semantics to software artifacts, which can be used by inference engines to provide new functionalities such as semantic retrieval, or suggestion of relevant knowledge. The SRS provides easy integration with client applications through the semantic web service. Thus, tools for Software Engineering and MDA (Model Driven Architecture) that use metadata (for exemple, the EODM plug-in for Eclipse) can be integrated in SRS has knowledge providers. These tools can also be modified to act as knowledge retrievers.

References

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

2. Zuniga, G.L.: Ontology: Its transformation from philosophy to information systems. In: Proceedings of the International Conference on Formal Ontology in Information Systems, ACM Press (2001) 187–197
3. Liebowitz, J., Wilcox, L.C.: Knowledge Management and Its Integrative Elements. CRC Press (1997)
4. McGuinness, D.L., van Harmelen, F.: Owl web ontology language overview. Technical report, W3C (2004)
5. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. IEEE Intelligent Systems **16**(2) (2001) 46–53
6. Martin, D.L., Paolucci, M., McIlraith, S.A., Burstein, M.H., McDermott, D.V., McGuinness, D.L., Parsia, B., Payne, T.R., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.P.: Bringing semantics to web services: The owl-s approach. In Cardoso, J., Sheth, A.P., eds.: Semantic Web Services and Web Process Composition, First International Workshop, SWSWPC 2004, San Diego, CA, USA, July 6, 2004, Revised Selected Papers. Volume 3387 of Lecture Notes in Computer Science., Springer (2004) 26–42
7. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the semantic web recommendations. Technical Report HPL-2003-146, HP Laboratories Bristol (2003)
8. Hatcher, E., Gospodnetic, O.: Lucene in Action. Manning Publications (2004)
9. Hillmann, D.: Using dublin core (2005)
10. Sametinger, J.: Software Engineering with Reusable Components. Springer (1997)
11. Miller, G.A.: Wordnet: A lexical database for english. Communications Of The ACM **38**(11) (1995) 39–41
12. Ding, L., Finin, T.W., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: A search and metadata engine for the semantic web. In Grossman, D., Gravano, L., Zhai, C., Herzog, O., Evans, D.A., eds.: Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004, ACM Press (2004) 652–659
13. Pinto, H.S., Gómez-Pérez, A., ao P. Martins, J.: Some issues on ontology integration. In: In Proceedings of the Workshop on Ontologies and Problem Solving Methods. (1999)
14. Alani, H.: Position paper: Ontology construction from online ontologies. In Carr, L., Roure, D.D., Iyengar, A., Goble, C.A., Dahlin, M., eds.: Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006, ACM (2006) 491–495
15. Jackson, P., Moulinier, I.: Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization. John Benjamins (2002)
16. Hyland-Wood, D., Carrington, D., Kaplan, S.: Toward a software maintenance methodology using semantic web techniques. In: Proceedings of Second International IEEE Workshop on Software Evolvability. (2006)
17. Seaborne, A., Prud'hommeaux, E.: Sparql query language for rdf. Technical report, W3C (2006)
18. Hyland-Wood, D.: An owl-dl ontology of software engineering concepts (2006)
19. S. Thaddeus, S.V.K.R.: A semantic web tool for knowledge-based software engineering. In: 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), Athens, G.A., USA, Springer (2006)