A Semantics-Based Approach to Software Reuse

Andrea Taglialatela[1], Francesco Taglino[2]

[1]Tecnologia nelle Reti e nei Sistemi - T.R.S. SpA
Via della Bufalotta 378
00139 Rome, Italy
andrea.taglialatela@trs.it
[2]Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti" - IASI-CNR
Viale Manzoni 30,
00185 Rome, Italy
francesco.taglino@iasi.cnr.it

Abstract. This work presents the ongoing development of a semantics-based method for supporting software reuse. Among the different paradigms the reuse of software can be characterized by, we focus on the retrieval aspect. To this end, the proposed method is based on three main aspects: (i) the building of a domain reference ontology. This is carried on through the adoption of the SOBE (Social Ontology Building and Evolution) method; (ii) semantic annotation of software artefacts. We introduced the concept of Semantic Descriptor as the mean for creating semantics-based information proxies of the software artefacts; (iii) semantics-based search and retrieval of software artefacts. We adopted SemSim, a semantic similarity matching method. The work is conducted within the Sentinel project.

1 Introduction

Software reuse is a fundamental aspect of industry best practices. It represents a degree of maturity of software development and provides clear benefits [12]:

- *Increased dependability*: reused software, which is tried and tested in working systems, should be more dependable than new software;
- *Reduced process risk*: if software exists, there is less uncertainty in the cost of reusing that software than in the cost of development;
- *Effective use of specialists*: instead of application specialists doing the same work on different projects, they can develop reusable software that encapsulate their knowledge;
- *Accelerated development*: reusing software can speed up system production because both development and validation time should be reduced.

However, software reuse is not a trivial practice, since it requires that development of software components follows clear characteristics [11]:

- *Portability*, to reduce the effort required to support applications across different OS platforms, programming languages, and compilers;

- *Flexibility*, to support a growing range of patterns;
- *Extensibility*, to support updates and additions of new features;
- *Reliability*, to ensure that software components are robust and fault tolerant.

In addition to that, software reusability has to be also supported by mechanisms for easily and efficiently access and retrieve available software artefacts. This work addresses this aspect, by presenting an ontology-based approach for creating a semantically enriched library of software artefacts for supporting software artefacts retrieval.

The proposed approach is based on the following steps:

- **Definition of the domain specific knowledge**: this is a crucial aspect in addressing any interoperability problem. Different actors and stakeholders (e.g., software developers and designers) need to share a common reference in order to act in an heterogeneous environment. In our case, this common reference is represented by a domain ontology which describes the relevant concepts the software talks about (e.g., air traffic control). To this end, we propose SOBE (Social Ontology Building Evolution) a method which mixes automatic knowledge extraction and social participation facilities for building domain ontologies.
- **Semantic description of software artefacts**: for each software artefact, a semantics-based image is built by instantiating a semantic descriptor (SD) which allows the description of a piece of software by using the common domain ontology.
- **Semantics-based search and retrieval of software artefacts**: once a repository of semantic descriptors has been established searching for software artefacts means to query semantic descriptors and retrieve software resources by following the link to the actual software repository. This allow people to interact with homogeneous proxies of the actual resources, since descriptors are characterized by content from the common domain ontology.

The paper is organized in the following way. In section 2, SOBE, the proposed method for ontology building is presented. In section 3, the description of the method for the semantic description of software artefacts is reported. In section 4, the semantic search similarity method for retrieval of semantically described software artefacts is described. Section 5 reports about related works, and finally Conclusions and Future Work section ends the manuscript.

2 SOBE: a method for social participation in building reference ontologies

The methodology for building the reference ontology representing the application domain that certain software components refer to is SOBE (Social Ontology Building and Evolution). It derives from the UPON [1] methodology.
UPON is a methodology for building domain ontologies developed along the guideline of the Unified Process [4], a consolidated method in the context of software engineering. UPON is based on an incremental process which is given by the enrichment of intermediate artefacts (e.g., a lexicon, and a glossary).
Inspired by UPON, SOBE is characterized by the following main aspects:

- An incremental approach through the production of the following results:
  - **Lexicon**, that defines the shared terminology;
  - **Glossary**, through the enrichment of the terms in the lexicon by identification of natural language descriptions. Once natural language definitions have been attached, synonyms can be identified. In the case of synonyms, one of the term is chosen as *preferred term*;
  - **IS-A taxonomy**, through the identification of generalization/specialization relationships between concepts;
  - Definition of further relationships, such as part-of.
- Knowledge extraction from documental resources: in building ontologies, especially when they are about very specific application domains (e.g., in the case of the air traffic control), it is necessary to refer to existing sources. Such sources can come either from standards (or somehow validated), or from the company developing the software (e.g., technical specifications, source code). The former contribute in giving solidity to the ontology, while the latter contribute in terms of specific company needs. The proposed methodology considers informative sources of different nature:
  - Non structured sources: for these typologies of textual documents natural language technique are used to analyze the text and extract terms and more complex linguistic expressions (chunk extraction);
  - Structured sources (e.g., glossaries or taxonomies): which are considered as they are.
- Social participation of domain experts involved in the application domain the software library refers to. This step aims at validating and enriching a consensus on the contents to be included in the ontology. The social participation to the ontology building is supported by two main aspects:
  - **Voting**: to allow domain experts to express preferences on the addition or removal of automatic extracted contents.
  - **Discussing**: to evaluate in a collaborative manner the introduction of certain contents in the ontology.

3 A method for semantic enrichment of software artefacts

With semantic enrichment we here intend the possibility to associate to a digital resource (in our case to a software artefact, such as, Java Classes or Packages) a description built in terms of a reference ontology.
The proposed methodology is characterised by the following aspects:

- Semantic Descriptor: the informative structure which allows the annotation to be represented;
- An automatic support to the annotation which is based on the analysis of comments accompanying the software code;
- Manual validation of the automatic suggestion.

3.1 Semantic Descriptor structure

The Semantic Descriptor represents the informative structure which represents the annotation of a single software artefact. The descriptor is organized in accordance with the following structure:
Header: collects the following information items:

- **Name**: the name of the artefact;
- **Description**: a natural language description of the artefact;
- **Author**: who developed the artefact;
- **Language**: programming language use in the implementation of the artefact;
- **Release Date**: date in which the artefact has been released;
- **ResourceLink**: a reference to the actual artefact. Following the value of this field it is possible from the descriptor to access to the actual resource described by the descriptor itself;

Some of these fields are taken from the Dublin core Vocabulary[1] used to describe documents.

Content: represents the actual semantic contribution of the descriptor. The Content section collects the set of concepts belonging to the built reference ontology which describe the artefact at best. This concepts collection is represented in the form of an Ontology-based Feature Vector (OFV) [3]. When it represents the annotation it is here called annotation ofv (*a-ofv*).

Documental references: allow references to documents (e.g., technical specifications, API documentation) to be specified in the semantic descriptor of the software artefact..

3.2 Automatic support to the semantic annotation of software artefacts

When we want to annotate a huge number of software artefacts (e.g., Classes), it is useful to have an automatic supporting mechanism giving annotation suggestions. The proposed methodology provides a mechanism like this which is based on the analysis of the text representing the comments added by the developer to the artefacts. This means that the result of the automatic support depends much on the quality of the comments in the code.
Given a software artefact (e.g., a Java class) the automatic support is organized in the following way:

– Identification of the information in the header of the Semantic Descriptor

  - Name: corresponds to the name of the artefact (e.g., the name of file or the folder);
  - Description: is given by the comment to the Class;

[1] http://dublincore.org/documents/dces/

    - Author: in a Java Class, it can be identified by a specific tag (@author) in the code, it it has been inserted;
    - Language: it can be identified by looking at the extension of the file

- Identification of the concepts from the ontology that describe the artefact (Content section):

    - from the code, comments to the Class and to each method are extracted;
    - comments are analysed by using natural language processing techniques in order to extract nouns in the text. Not only single-word terms are extracted, but also multi-words ones.

- In order to identify concepts that are suitable for describing an artefact, extracted terms are matched with the concepts in the reference ontology. For this purpose, string matching techniques are applied. If a term extracted from the comments corresponds to the label of a concept or a synonym of its, then the concept is inserted in the Content section. If the term occurs in the natural language description of a concept in the reference ontology, then the concept is inserted in the Content section but marked as "Suspected".

3.3 Manual validation of the automatic suggestion

The automatic support represents a suggestion and as such, it needs to be validated by a human being who can modify, add or remove proposed annotations. In particular, those concepts that have been inserted in the Content section as *Suspected* should have analysed accurately.

4 Semantic search of software artefacts

Once software artefacts have been semantically annotated, search for them can be performed on semantic descriptors without inspecting software artefacts directly. Indeed, semantic descriptors have a defined structure and their content is strongly characterized by the reference domain ontology. In particular, being enriched by concepts from the reference ontology, allows us to apply a semantic similarity search method which exploit the Content section of the semantic descriptor.

The schema of a query is represented by the same structure of a semantic descriptor in terms of both Header and Content sections.

About the criteria related to the Header section (e.g., the author or the programming language), an exact matching is required. For what concerns search criteria related to the Content section, a similarity metrics exploiting the information content of the concepts in the ontology and their specialization/generalization relationships is applied. In particular, a request ofv (*r-ofv*) is specified. For each software artefact, the semantic similarity method (*SemSim*) [3] computes the semantic similarity between the *a-ofv* associated to the artefact and the *r-ofv*. On the basis of the obtained similarity degree, a ranking of the software artefacts is defined.

Since the *SemSim* method is based on an information content approach, each concept in the ontology has associated a weight representing the probability that the concept annotates a software artefact. In this case we talk about a Weight Reference Ontology (WRO) [3].

Finally, the search method is based on three steps:

- Given an *a-ofv* and a *r-ofv*, for each pair of concepts (c_i, c_j), where c_i belongs to *a-ofv* and c_j belongs to *r-ofv* is computed. The semantic similarity between pairs of concepts is computed on the basis of their information content [5], [10];
- the similarity between each *a-ofv* and the *r-ofv* is computed in accordance with the maximum weighted matching problem in bipartite graphs [2];
- The answer is given in terms of a list of software artefacts on the basis of the similarity degree corresponding to the *a-ofv* in the related semantic descriptor.

5 Related work

[8] groups the approaches for search and retrieval of software artefacts into four categories: simple keyword-based text search, faceted classification and retrieval, signature matching and behavioural matching.

Keyword-based searching is the simplest approach and its effectiveness depends on the names of the components themselves [7].

Faceted classification and retrieval involves extracting keywords from components description and documentation and arranging this information into a predefined classification scheme (e.g., a taxonomy). Despite promising results [9], this approach is very effective if maintenance to the classification scheme is provided in efficient way.

Pure signature-based matching approaches, like [6], describes components on the basis of their input and output parameters, but components having matching signatures do not guarantee to be related.

Behavioural matching extends signature matching and attempts to describe the behaviour of a component. However, this approach appears cumbersome and hard to apply [13].

Among semantics-based approaches we recall Fusion [14] which provides a method for describing components similar to the one used for describing services. However, it allows only one single domain category to be specified for describing a software component.

6 Conclusions and Future Work

In this paper we presented a semantics-based approach aimed at supporting the re-use of software artefacts. The approach is articulated in three main steps: (i) building an ontology related to the specific domain the software talks about; (ii) annotating the software artefacts. The annotation is performed instantiating semantic descriptors that represent proxies of the software artefacts and describe their content in terms of the domain ontology; (iii) allowing the search and retrieval of software artefacts by querying the repository of the semantic descriptors. The search functionality is characterised by semantic similarity reasoning that works on the domain specific description of the software artefacts.

As future work, we intend to enrich the semantic similarity search by explaining, in a human interpretable way, the meaning of the returned results, that are currently shown with a corresponding similarity degree. Furthermore, in line with the objectives of the Sentinel project, we intend experiment the presented approach in a real case that will be related to the air traffic control domain.

In addition, we also intend to extend the annotation criteria by taking into consideration structural aspects of the software artefacts, such as dependencies among software components.

References

1. De Nicola, A., Navigli, R., Missikoff, M. A software engineering approach to ontology building, Information Systems, 34 (2), 258-275, 2009.
2. Formica, A., Missikoff, M., 2002. Concept Similarity in SymOntos: an Enterprise Ontology Management Tool. Computer Journal 45(6), 583--594 (2002).
3. Formica, A., Missikoff, M., Pourabbas, E., Taglino, F., 2010. Semantic Search for Enterprises Competencies Management. In the Proc. Of the International Conference on KNowledge Engineering and Ontology Development (KEOD), 2010.
4. Jacobson I., Booch G., Rumbaugh J. The Unified Software Development Process, Addison Wesley, 1999.
5. Lin, D., 1998. An Information-Theoretic Definition of Similarity. In proc. of 15th the International Conference on Machine Learning. Madison, Wisconsin, USA, Morgan Kaufmann, 296—304. Shavlik J. W. (ed.).
6. Luqi, Guo, J., 1999. Toward Automated Retrieval for a Software Component Repository. In Proc, of the 6th Symposium on Engineering of Computer-Based Systems (ECBS 99).
7. Mili, R., Mittermeir, R.T., 1994. Storing and Retrieving Software Components: a Refinement Based System. In Proc. Of the 16th International Conference of Software Engineering.
8. Mili, R., Mili, A., Mittermeir, R.T., 1998. A Survey of Software Storage and Retrieval, Annals of Software Engineering, vol. 5, no. 2, pp. 349-414.
9. Ostertag, E., Hendler, J., Prieto-Diaz, R., Braun, C., 1992. Computing Similarity in a Reuse Library System – An AI Approach. ACM Transactions on Software Engineering and Methodology, vol. 1, no. 3.
10. Resnik, P., 1995. Using information content to evaluate semantic similarity in a taxonomy. In proc. of IJCAI.
11. Schmidt, D., 1999. Why Software Reuse has Failed and How to Make it Work for You. C++ Report, Vol 11, Issue 1, 1999.
12. Sommerville, I., 2007. Software Engineering. Pearson Education, 2007.
13. Sugumaran, V., Storey, V.C., 2003. A Semantic-Based Approach to Component Retrieval. SIGMIS Database, vol. 34, no. 3.
14. Zygkostiotis, Z., Dranidis, D., Kourtesis, D., 2009. Semantic Annotation, Publication, and Discovery of Java Software Components: An Integrated Approach. In Proc of the Work-

shops of the 5th IFIP Conference on Artificial Intelligence Applications & Innovations (AIAI-2009).