Enhanced Semantic Access to Software Artefacts

Danica Damljanović and Kalina Bontcheva

Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello Street
S1 4DP, Sheffield, UK
{D.Damljanovic,K.Bontcheva}@dcs.shef.ac.uk

Abstract. Large software frameworks and applications tend to have a significant learning curve both for new developers working on system extensions and for other software engineers who wish to integrate relevant parts into their own applications. Recent research has begun to demonstrate that semantic technologies are a promising way to address some of these issues. In this paper, we present a semantic-based prototype that is made for an open-source software engineering project with the goal to explore the methods for assisting open-source developers and software users to learn and maintain the system without major effort.

Key words: semantic annotation, ontology learning, semantic access, software artefacts

1 Introduction

Successful code reuse and bug avoidance in software engineering requires numerous qualities, both of the library code and of the development staff; two important qualities are ease of identification of relevant components and ease of understanding of their parameters and usage profiles.

The attraction of using semantic technology to address this problem lies in its potential to transform existing software documentation into a conceptually organised and semantically interlinked knowledge space that incorporates unstructured data from multiple software artefacts: forum postings, manuals, structured data from source code and configuration files. The enriched information can then be used to add novel functionality to web-based documentation of the software concerned, providing the developer with new and powerful ways to locate and integrate components (either for reuse or for integration with new development).

In the context of the TAO (`tao-project.eu`), we have developed a semantic-based prototype, on the basis of GATE (`gate.ac.uk`) – widely used open-source software project. The goal of this prototype is to explore the methods for assisting distributed, dynamic groups of software developers and users to learn and maintain this system without major effort, through the application of semantic web technologies. As the core of any semantic-enabled system is in ontologies, we first

acquired the domain ontology semi-automatically from the GATE source code, documentation, manuals and other software artefacts. The domain ontology is used for the semantic content augmentation process, to annotate automatically all software artefacts. The results are stored in a semantic annotation repository to enable users to carry out semantic searches and easily find all information relevant to a given GATE concept.

The paper is structured as follows. In Section 2 we discuss requirements for the GATE case study. In order to meet these requirements, we developed the semantic-enabled prototype which is described in Section 3. Section 4 draws conclusions and outlines directions for future work.

2 The Case Study

GATE [1] is an open-source, general architecture for text engineering, used by thousands of users at hundreds of sites. The development team consists at present of over 15 people, but over the years more than 30 people have been involved in the project. As such, this software product exhibits all the specific problems that large long-running open-source projects encounter.

While GATE has increasingly facilitated the development of knowledge-based applications with semantic features (e.g. [2–4]), its own implementation has continued to be based on functionalities justified on the syntactic level, understood by informal human-readable documentation. By its very nature as a successful and accepted 'general architecture', a systematic understanding of its concepts and their relation is shared between its human users. It is simply that this understanding has not been formalised into a description that can be reasoned about by machines or made easier to access by new users. Indeed, novice GATE users are finding it difficult due to the large amount of heterogeneous information, which cannot be accessed via a unified interface.

Using machine-understandable language to interpret facts about GATE means using ontologies to transform existing software documentation (user manuals, source code, forum posts) into a conceptually organised and semantically interlinked knowledge space. Such a knowledge space could be a step towards enhanced knowledge access to support distributed teams of software developers and users. In order to build and query such a knowledge space, following needs to be done:

- develop domain ontology based on software artefacts,
- implement a semantic annotation process which indexes software artefacts regularly with respect to the domain ontology and updates the knowledge base/semantic annotation repository,
- enable accessing the knowledge base in a user-friendly manner.

In the next section, we present a semantic-based prototype system for enhanced access to software artefacts developed in order to meet the requirements above. The core of this prototype is in the domain ontology, which has been designed semi-automatically using ontology learning tools. Due to the space

limitations we will not detail learning domain ontology from software artefacts here, as that is explained elsewhere (see [5]).

3 The Semantic Annotation and Knowledge Access Prototype

In order to collect software artefacts about GATE which are dispersed across different locations on the Web, we implemented a crawler which is downloading relevant data (Section 3.1). These data are then being processed by the *CA Service* for semantic annotation (Section 3.2). Automatically produced annotations are exported by the *CA (Content Augmentation) Index* (Section 3.3) and stored in the *knowledge store*. Finally, these annotations are made accessible through text-based queries (Section 3.4).

3.1 Data Collection

Gathering relevant data about GATE required implementing a crawler which visits `gate.ac.uk` and downloads manuals, JavaDoc, source code, papers (including those from external links) and other software artefacts. Additionally, the crawler is visiting the GATE mailing list which is hosted on `sourceforge.net`, and downloading all new posts, which have not been indexed already in previous iterations (see Figure 1). This process is run on a daily basis to capture and index new software artefacts, as soon as they become available. At the time of writing we have collected around 2GB of content (10000 documents), the majority of which is text-based.

When downloading documents, we not only store their content (docContent in Figure 1) but also the URL from which the document was downloaded (docURL) and the type of the document (docType). We use several simple heuristic rules in order to predict what is the type of the document based on the URL. For example, if the URL contains javadoc, it is easy to conclude that the document is a source documentation file. Other types are: paper, forum post, Web page, and source code. Once all software artefacts are downloaded and stored, they need to be enriched with semantic information. In our case, that process is performed automatically, as explained next.

3.2 Automatic Content Augmentation

For annotation purposes we use the CA (Content Augmentation) Service (see Figure 1) which wraps Key Concept Identification Tool (KCIT). KCIT is an Information Extraction application, based on several general-purpose GATE [6] components plus an ontology-based gazetteer which is capable of producing ontology-aware annotations automatically, i.e., annotations referring to classes, instances and properties in the ontology [7].

The output of the semantic annotation process is a set of annotations and their features: the *URI* of the ontology resource to which the term refers to, its

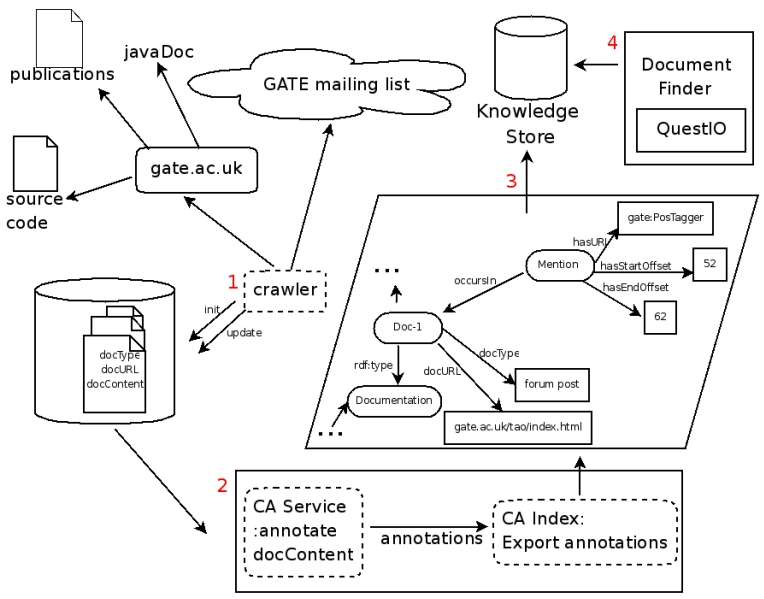


Fig. 1. System architecture

type (e.g., an instance, a class, or a property), and other features that could be used later during search. An example of a semantically annotated document (a Java class from the GATE source code) is shown in Figure 2. The pop-up table depicts annotation features created by KCIT for the annotated term 'Niraj Aswani'. From these features, it can be concluded that this name is referring to a GATE developer as, according to the features, this name is a value (*propertyValue*) of the property *rdfs:label* (*propertyURI*) for an instance (*type*) that is of type *GATE developer* (*classURI*). Value 0 for *heuristic level* indicates that no heuristic rules were used during the semantic annotation process.

Once the semantic content augmentation stage is completed, document annotations needs to be merged with document metadata (docURL and docType) and saved in a way that makes them accessible through semantic search.

3.3 Storing Implicit Annotations

The annotation extraction phase (performed via the *CA Index* shown in Figure 1) comprises of reading produced annotation features, merging them with document-level metadata, and exporting them in a format which is then easily queried via a formal language such as SPARQL. More specifically, this extracted information needs to 'connect' a *document* with different *mentions* of the ontology resources inside that document. For example, if a document contains mentions of the class *Sentence Splitter*, the output should be modeled in a way that preserves this information during query time (i.e. the URLs of all documents mentioning this class should be found easily). For this purpose, we use

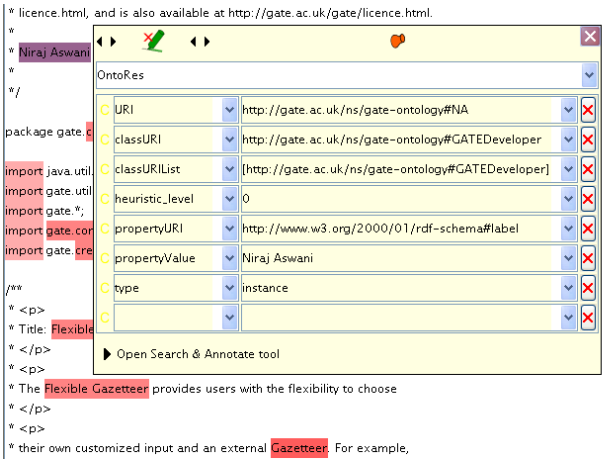


Fig. 2. Annotating `FlexibleGazetteer.java` class with KCIT

the PROTON KM ontology[1], and more specifically, the *Mention* and *Document* classes.

The *Document* class has several properties defined, among which we use: *resourceType* (refers to the type of the document) and *informationResourceIdentifier* property (refers to the URL of the annotated document). In the example of the extracted OWL output, generated from the annotated document shown in Figure 2, the *Document* class is instantiated as follows:

```
<rdf:Description rdf:about=
 "gate:id_ee7ba66b-cd71-4993-9635-777b24f46372">
<protont:informationResourceIdentifier>
  http://gate.ac.uk/gate/doc/java2html/gate/creole/gazetteer/
  FlexibleGazetteer.java.html
</protont:informationResourceIdentifier>
<protonkm:resourceType>  Source Code   </protonkm:resourceType>
</rdf:Description>
```

For the *Mention* class, defined properties for storing the position of the semantic annotation within the document content are used, namely *hasStartOffset* and *hasEndOffset*. Property *occursIn* links the two classes, *Mention* and *Document*.Property *refersAnything* is newly defined in order to preserve the URI of the resource to which a *Mention* is referring to.

An example instance of *Mention* and its relation to the above mentioned instance of *Document* is encoded as follows:

```
<rdf:Description rdf:about=
"gate:mention_0c45b1dc-efab-48a2-8242-bb78c1ddd3b5">
<rdf:type rdf:resource=
"http://proton.semanticweb.org/2005/04/protonkm#Mention"/>
```

[1] `http://proton.semanticweb.org/2005/04/protonkm`

```
<protonkm:occursIn rdf:resource=
"gate:id_ee7ba66b-cd71-4993-9635-777b24f46372"/>
<protonkm:hasStartOffset>  404   </protonkm:hasStartOffset>
<protonkm:hasEndOffset>  409  </protonkm:hasEndOffset>
<gate:refersAnything
rdf:resource="  http://gate.ac.uk/ns/gate-ontology#NA"/>
</rdf:Description>
```

Note that *gate:* is used in the examples above instead of the full namespace of the ontology which is `http://gate.ac.uk/ns/gate-ontology#` simply for the sake of brevity. The long names for the new instances of both *Document* and *Mention* classes are created automatically.

The extracted annotations are stored in an OWL-compatible knowledge repository (OWLIM [8]), and accessible for querying using formal query languages (e.g.,SeRQL, SPARQL). Such languages – while having a strong expressive power – require detailed knowledge of their formal syntax and understanding of ontologies. One of the ways to lower the learning overhead and make semantic-based queries more straightforward is through a text-based queries.

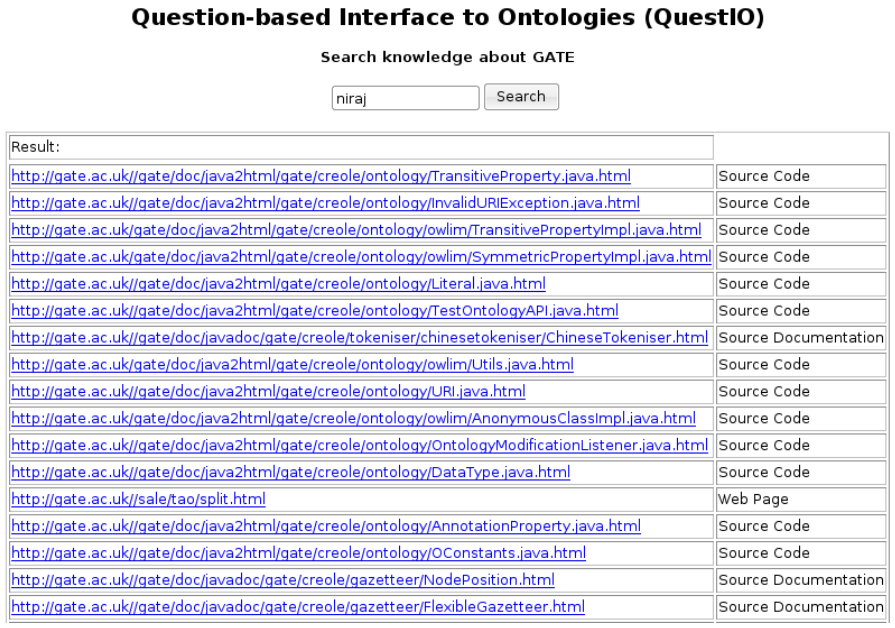
3.4 Semantic-based Access through Text-based Queries

In order to enable advanced semantic-based access through text-based queries, we have customised a Question-based Interface to Ontologies – QuestIO (Document Finder in Figure 1), which we have developed in our previous work [7, 9]. QuestIO is a domain-independent system which translates text-based queries into the relevant SeRQL queries, executes them and presents the results to the user. QuestIO works so that it first recognises key concepts inside the query, detects any potential relations between them, and creates the required semantic query. For example, if the query consisted of two concepts (e.g. 'plugins in GATE') the matching triples from the ontology will be extracted and shown (in this case – a list of all instances of GATE plugins).

In order to access the data stored in the implicit annotations (i.e. the URLs of *Documents* with particular *Mentions*) we had to make QuestIO more intuitive, by customising it so that the user can omit some obvious concepts when posting the query. For example, if the user needs more information about the *Sentence Splitter parameters* (i.e. doc URLs which mention these concepts), the query for QuestIO would need to be formed as *documents about Sentence Splitter parameters*. We customised QuestIO so that *document* is added to the query by default, so that users do not have to specify this explicitly each time.

Also, as the output of QuestIO is a set of triple-like rows, we have customised it to produce a two column table of results (the first column showing document URLs, the second showing document types), rather than a table with relations between concepts. An example query with results is shown in Figure 3. For the query 'niraj', list of documents mentioning this term is returned, among which the last link points to the documentation about Flexible Gazetteer. This is inline with the Figure 2, from which it can be concluded that Niraj is the author of the class *FlexibleGazetteer.java*. The advantage of the *semantics* used

in the prototype is such that queries are observed as concepts, not like a set of characters – as it is the case in traditional search engines. For example, *Niraj*, *Niraj Aswani*, or *NA* (as initials) would all return the same results as soon as the ontology encodes that these terms refer to the one particular concept.



Question-based Interface to Ontologies (QuestIO)

Search knowledge about GATE

niraj Search

Result:	
http://gate.ac.uk//gate/doc/java2html/gate/creole/ontology/TransitiveProperty.java.html	Source Code
http://gate.ac.uk//gate/doc/java2html/gate/creole/ontology/InvalidURIException.java.html	Source Code
http://gate.ac.uk/gate/doc/java2html/gate/creole/ontology/owlim/TransitivePropertyImpl.java.html	Source Code
http://gate.ac.uk/gate/doc/java2html/gate/creole/ontology/owlim/SymmetricPropertyImpl.java.html	Source Code
http://gate.ac.uk//gate/doc/java2html/gate/creole/ontology/Literal.java.html	Source Code
http://gate.ac.uk//gate/doc/java2html/gate/creole/ontology/TestOntologyAPI.java.html	Source Code
http://gate.ac.uk//gate/doc/javadoc/gate/creole/tokeniser/chinesetokeniser/ChineseTokeniser.html	Source Documentation
http://gate.ac.uk/gate/doc/java2html/gate/creole/ontology/owlim/Utils.java.html	Source Code
http://gate.ac.uk//gate/doc/java2html/gate/creole/ontology/URI.java.html	Source Code
http://gate.ac.uk/gate/doc/java2html/gate/creole/ontology/owlim/AnonymousClassImpl.java.html	Source Code
http://gate.ac.uk//gate/doc/java2html/gate/creole/ontology/OntologyModificationListener.java.html	Source Code
http://gate.ac.uk//gate/doc/java2html/gate/creole/ontology/DataType.java.html	Source Code
http://gate.ac.uk//sale/tao/split.html	Web Page
http://gate.ac.uk//gate/doc/java2html/gate/creole/ontology/AnnotationProperty.java.html	Source Code
http://gate.ac.uk//gate/doc/java2html/gate/creole/ontology/OConstants.java.html	Source Code
http://gate.ac.uk//gate/doc/javadoc/gate/creole/gazetteer/NodePosition.html	Source Documentation
http://gate.ac.uk//gate/doc/javadoc/gate/creole/gazetteer/FlexibleGazetteer.html	Source Documentation

Fig. 3. List of results for the query 'niraj'

At the moment, our prototype is returning a list of all relevant documents, without any ranking. In future work we will investigate methods for result summarisation and clustering.

4 Conclusion and Future Work

This paper described a prototype for enhanced semantic access to software artefacts using the GATE open-source project as an example. In contrast to approaches such as OSEE [10], we do not alter the software development practices, but rather layer some semantic technology on top, to enable new usage of already existing software artefacts. In this respect, our work is similar to the Dhruv bug resolution system [11], which, unlike us, however encountered scalability problems with the semantic repository and also did not examine ontology learning as a way of bootstrapping the process.

Our approach consists of three basic steps. Firstly, the domain ontology is either authored manually or bootstrapped through ontology learning and population techniques. The second phase is semantic annotation which is performed fully automatically. The generated annotations together with document metadata are stored in a repository in OWL format and are made accessible via natural language-based queries.

Our future work will focus on the improvement of the current interface, and the implementation of result clustering and summarisation. For the evaluation of the prototype, in the forthcoming months we will carry out a user-centric evaluation which will be along the following dimensions:

- finding the specific information, with and without semantic-based access,
- benefits and usability of our language-based knowledge access approach,
- scalability of the knowledge stores and ability to store all software artefacts.

Acknowledgements. This research was partially supported by the EU Sixth Framework Program project TAO (FP6-026460).

References

1. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). (2002)
2. Bontcheva, K., Tablan, V., Maynard, D., Cunningham, H.: Evolving GATE to Meet New Challenges in Language Engineering. Natural Language Engineering **10**(3/4) (2004) 349—373
3. Kiryakov, A., Popov, B., Ognyanoff, D., Manov, D., Kirilov, A., Goranov, M.: Semantic annotation, indexing and retrieval. Journal of Web Semantics, ISWC 2003 Special Issue **1**(2) (2004) 671–680
4. Sabou, M.: Building Web Service Ontologies. PhD thesis, Vrije Universiteit (2006)
5. Bontcheva, K., Sabou, M.: Learning Ontologies from Software Artifacts: Exploring and Combining Multiple Sources. In: Workshop on Semantic Web Enabled Software Engineering (SWESE), Athens, G.A., USA (November 2006)
6. Cunningham, H.: GATE, a General Architecture for Text Engineering. Computers and the Humanities **36** (2002) 223–254
7. Damljanovic, D., Tablan, V., Bontcheva, K.: A text-based query interface to owl ontologies. In: 6th Language Resources and Evaluation Conference (LREC), Marrakech, Morocco, ELRA (May 2008)
8. Kiryakov, A.: OWLIM: balancing between scalable repository and light-weight reasoner. In: Proc. of WWW2006, Edinburgh, Scotland (2006)
9. Tablan, V., Damljanovic, D., Bontcheva, K.: A natural language query interface to structured information. In: Proceedings of the 5h European Semantic Web Conference (ESWC 2008), Tenerife, Spain (June 2008)
10. Thaddeus, S., Raja, S.K.: A Semantic Web Tool for Knowledge-based Software Engineering. In: Workshop on Semantic Web Enabled Software Engineering (SWESE), Athens, G.A., USA (2006)
11. Ankolekar, A., Sycara, K., Herbsleb, J., Kraut, R.: Supporting Online Problem Solving Communities with the Semantic Web. In: Proc. of WWW. (2006)