Semantic Web 1 (2009) 1–5
IOS Press

Transition of Legacy Systems to Semantically Enabled Applications: TAO Method and Tools

Editor(s): Krzysztof Janowicz, Pennsylvania State University, USA
Solicited review(s): Patrick Maué, University of Münster, Germany; Todd Pehle, Orbis Technologies, USA

Hai H. Wang [a,*], Danica Damljanovic [d], Terry Payne [b], Nicholas Gibbins [c], and Kalina Bontcheva [d]
[a] *Aston University, Birmingham, UK, E-mail: H.WANG10@aston.ac.uk*
[b] *University of Liverpool Liverpool, UK, E-mail: T.R.Payne@liverpool.ac.uk*
[c] *University of Southampton, UK, E-mail: nmg@ecs.soton.ac.uk*
[d] *University of Sheffield, UK, E-mail: {D.Damljanovic, bontcheva}@dcs.shef.ac.uk*

Abstract. Despite expectations being high, the industrial take-up of Semantic Web technologies in developing services and applications has been slower than expected. One of the main reasons is that many legacy systems have been developed without considering the potential of the Web in integrating services and sharing resources. Without a systematic methodology and proper tool support, the migration from legacy systems to Semantic Web Service-based systems can be a tedious and expensive process, which carries a significant risk of failure. There is an urgent need to provide strategies, allowing the migration of legacy systems to Semantic Web Services platforms, and also tools to support such strategies. In this paper we propose a methodology and its tool support for transitioning these applications to Semantic Web Services, which allow users to migrate their applications to Semantic Web Services platforms automatically or semi-automatically. The transition of the GATE system is used as a case study.

Keywords: Semantic Web Services, Annotation

1. Introduction

Semantic Web (SW) and Semantic Web Service (SWS) technologies [19] have been recognised as very promising emerging technologies that exhibit huge commercial potential and have attracted significant attention from both industry and the research community [2]. Despite this promise, the resulting industrial take-up of SW and SWS technologies has been slower than expected. This is mainly due to the fact that many legacy systems have been developed without considering the potential of the Web for integrating services and sharing resources. The migration of legacy systems into semantic-enabled environments involves many recursive operations that have to be executed with rigour due to the magnitude of the investment in systems, and the technical complexity inherent in such projects. In this context, there are three main issues to be considered, namely: 1) *Web Accessibility*, dealing with the transformation of components of a legacy system that are exposed as Web services; 2) *Service Transformation*, where the exposed Web services are mapped to the corresponding Semantic Web Service representations; and 3) *Semantic Annotation*, where the service representations and software artefacts are annotated using the relevant domain ontology. Without a systematic methodology and proper tool support, the migration from legacy systems to semantically enabled applications could be a very tedious and expensive process, which carries a definite risk of failure. There is

*Corresponding author. E-mail: H.WANG10@aston.ac.uk

an urgent need to therefore provide strategies that support the construction of ontologies which facilite the migration of legacy systems to Semantic Web Services platforms, and also tools to support such strategies.

This paper reports on a new methodology and the related tool support for addressing the above issues, which in turn could lead to an automatic platform transformation. This work is part of the *Transitioning Applications to Ontologies (TAO)* project[1], which was part of the European Sixth Framework Program. In TAO, we created an open source infrastructure to aid transitioning of legacy applications to ontologies, through automatic ontology bootstrapping, semantic content augmentation, and generation of Semantic Web service descriptions. The work is grounded in the TAO transitioning methodology and the tool – *TAO Suite*. In this way, TAO enables a much larger group of companies to exploit semantics without having to re-implement their applications. All the related materials about TAO (e.g., the TAO softwares, source code, manuals, demos and deliverables) are publicly available at `http://www.tao-project.eu/`.

The remainder of this paper is organised as follows. Section 2 presents a set of cookbook-style guidelines for the TAO methodology and the usage of the supporting tools. Section 3 discusses the evaluation of the work. Finally Section 4 and 5 present the related work, conclusions of this paper and future work.

2. Methodology cookbook, tool support and case study

The methodology presented in this section provides a detailed view of the important phases to be performed as part of the transition process of legacy systems to semantic-enabled applications following the TAO scenario. It is fully supported by the TAO Suite, which is integrated from several software components. Figure 2 presents the architecture of the transitioning environment. In it, the *ontology learning tool* is used to derive an ontology from legacy application documentation (specifications, UML diagrams, code documentations, software manuals, including images). The *content augmentation tool* automatically identifies key concepts within legacy contents, which can go beyond textual sources, and annotates them using the domain ontology concepts. The distributed heterogeneous knowledge repositories are developed to efficiently index, query, and retrieve legacy content (including code, documentation, transcripts taken from discussion forums, etc), domain ontology and semantic annotations. An Integrated Development Environment (IDE) has been developed to provide an one-stop transition support for users.

One important novelty for the TAO transitioning methodology is that it provides a logical approach for connecting the traditional ontology and service design through the following main points.

- *Learning ontologies from service descriptions.* In a normal ontology design lifecycle, the *Ontology Learning* process attempts to automatically or semi-automatically derive a knowledge model from a document corpus. In our transitioning methodology, we refine and extend this to emphasis the contribution made by the description of a broader and more heterogeneous collection of documentation resources that relate to existing legacy applications (including application APIs, developer documentation, SOA design documentation, etc). We call this refinement *Service-Oriented Ontology Learning*.
- *Using domain ontologies to augment semantic content and service descriptions.* The service annotation process described in many existing SOA design methodologies refers to the description of services at the signature level in languages such as WSDL. While these allow rudimentary service matchmaking and brokerage on the basis of the types of the inputs and outputs of a service, these types are typically expressed syntactically using traditional data-types, rather than exploiting a semantically richer and more expressive representation grounded by an ontological characterisation of the relevant domain. Thus, these interfaces need to be mapped to equivalent concepts within Semantic Web frameworks (such as OWL-S, WSMO or WS-WSDL) and annotated using the relevant domain ontologies.

In the next subsection, we present a set of cookbook-style guidelines for the usage of TAO tools. Note that this paper only focuses on the usage of TAO tools, due to the limited space[2].

[1] `http://www.tao-project.eu/`

[2] For more technical details on various tool components, please refer to the respective reports, which can be downloaded from `http://www.tao-project.eu/`.

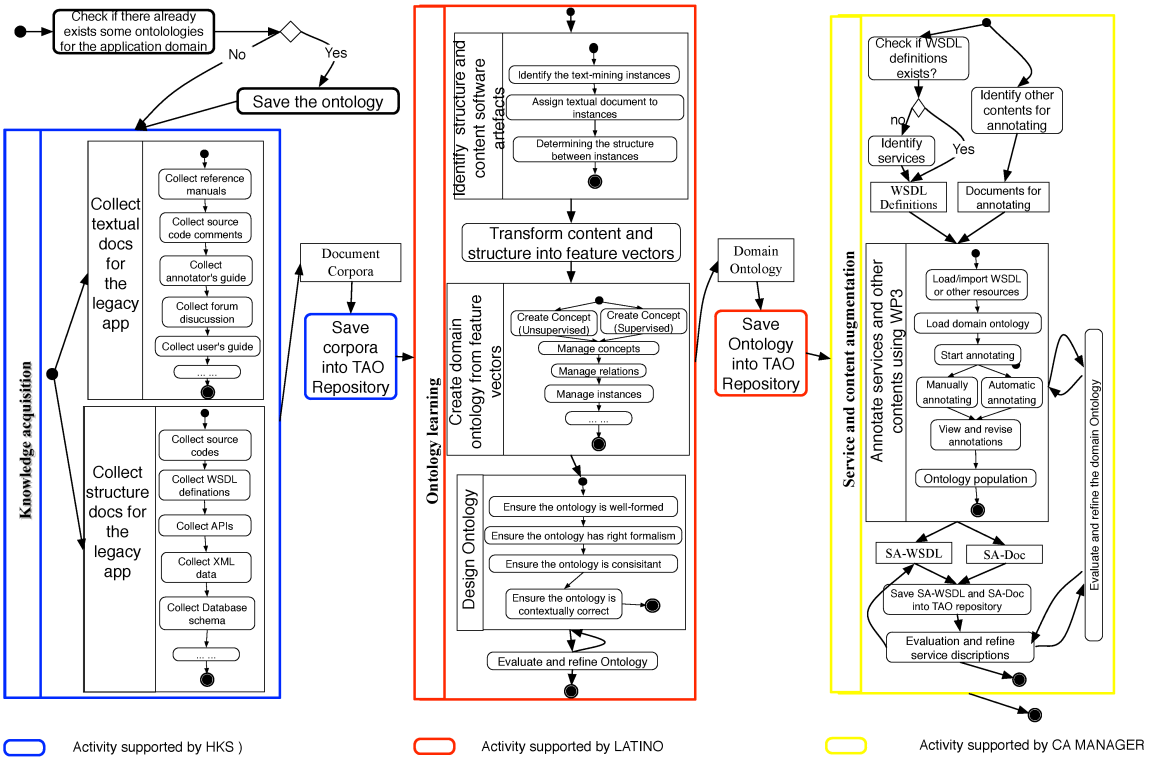


Fig. 1. Cookbook methodology overview.

To better illustrate the idea, we use as a case study the transition of the GATE[3] system to a collection of semantic-enhanced services. GATE is a leading open-source architecture and infrastructure for the building and deployment of *Human Language Technology* applications, used by thousands of users at hundreds of sites. After many years of developing, revising and extending, GATE developers and users find that it becomes difficult to understand, maintain and extend the system in a systematic way, due the large amount of heterogeneous information that cannot be accessed via a unified interface [3].

The advantages of transitioning GATE to semantic-enhanced services are two-fold. Firstly, GATE components and services will be easier to discover and integrate within other applications due to the use of Semantic Web Service technology. Secondly, the transition should facilitate better search results for queries over a given GATE concept due to the enhanced knowledge based searches that span a broader set of heterogeneous sources and corpora including the complete GATE document corpus, XML configuration files, video tutorials, screen shots, user discussion forum, etc. The development team of GATE consists at present of over 15 people, but over the years more than 30 people have been involved in the project. To be used for evaluating TAO tools, GATE exhibits all the specific problems that large software architectures encounter, which enables us to evaluate the methodology and tools intensively. Bontcheva et. al. [3] discuss the advantages and possibilities arising from building domain ontology and application for semantic enrichment of software artefacts in the GATE case study.

[3] http://gate.ac.uk

2.1. *Transitioning cookbook*

The TAO methodology has three main phases: the data acquisition phase, the ontology learning phase and the semantic content and service augmentation phase. Each phase contains a set of tasks which may interact with each other. Figure 1 presents a UML diagram that illustrates the main transitioning process; details of the major activities are presented below.

To transition a legacy application to a number of semantically enabled services, a software engineer should first check whether previously developed on-

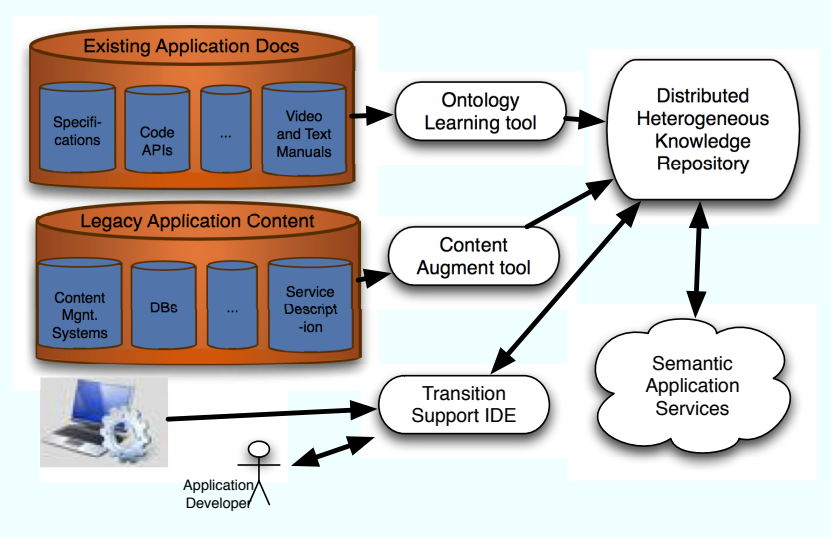


Fig. 2. Transitioning Process

tologies exist for the application domain. Public ontology search engines or ontology libraries may be used, such as Swoogle[4] or SWSE[5]. If no such ontology is found, users have to derive the domain ontology from the legacy software. If a related ontology is found, previous methodologies, such as the NEON methodology[6], could be used to directly adapt and extend the discovered ontology. However, given past experience in ontology development, these approaches are not ideal for many use cases. The main reason is that in most cases, it is difficult to find an existing ontology that is perfectly matched to the requirements of annotating and describing legacy applications. Reusing complex domain ontologies built for the domain in similar projects could be a tedious task. The more complex an ontology, and the more tied it is to its original context of development and use, the less likely it will fit another context. It can be as difficult and costly to trim such ontologies in order to keep only the relevant parts as it is to re-build those parts completely.

Building domain ontologies with a "top-down" approach as extensions of a "foundational ontology" is another popular approach. Foundational ontologies are often highly abstract, or include strong constraints that are rarely part of the requirements of the target system. In the TAO approach, if a related ontology is found, it is saved into the knowledge store developed by TAO and used as training data for the ontology learning tool together with other software artefacts. For the GATE case study, the ontology was developed from scratch with the assistance of the TAO tools.

2.1.1. Data acquisition

To derive the domain ontology from a legacy application using the TAO tools, users first need to collect relevant resources about the legacy application.

– Resources collection

In the TAO cookbook, we have identified various data sources which are commonly relevant to the description of a legacy application, such as application source code, APIs, and JavaDocs[7]. For the GATE case study, the application's Java source code and corresponding JavaDoc files are identified and assembled[8]. These are then deposited in the the TAO repository [24] – a component of the TAO tools.

– Save the resource corpora to the TAO Repository

The TAO repository [24] is a heterogeneous knowledge store designed and developed for efficient management of different types of knowledge: unstructured content (documents), structured data (databases), ontologies, and semantic annotations, which augment the content with links to machine-interpretable metadata. The query and reasoning capabilities of this repository are based on the Ontology Representation and Data Integration (ORDI) framework [24].

[4] http://swoogle.umbc.edu
[5] http://swse.deri.org
[6] http://www.neon-project.org/

[7] For more information about the potential data sources which may be related to the description of legacy systems and their classification, please refer to the report [1].

[8] Those documents can be downloaded from http://gate.ac.uk/download/index.html.

2.1.2. Ontology Learning

The process of ontology learning from software artefacts is essentially one of discovering concepts and relations from various learning resources identified in the previous steps, including the source code, accompanying documentation, and external sources (such as the Web). Ontology learning is one of the most significant approaches proposed to date for developing ontologies from existing domain-related resources. Previously, we have presented a detailed review of different ontology learning approaches [1]. Gomez-Perez et. al. [9] also reviewed the major methods for semi-automatically building ontologies from texts.

Two learning components were used for ontology learning within this case study: *LATINO* was used to build feature vectors from documentation; and *OntoGen*, was used to build an ontology from the resulting feature vectors. LATINO[9] is a general data-mining framework that unites text mining and link analysis for the purpose of (semi-automated) ontology construction. LATINO is novel with respect to other existing ontology learning methods in several ways. LATINO constructs the ontologies from implicit knowledge contained within the documents and data that form the set of learning resources (identified in the previous stages). Concepts and their inter/intro-relationships are selected and used in the ontology construction process. We introduce the term "application mining" to denote the process of extracting this knowledge from application-related resources. In addition, LATINO is not only limited to textual data sources; additional data resources that can be used for ontology learning, including structured documents such as database schema, UML models, existing source code, APIs, etc., or textual documents that include requirement documents, manuals, forum discussions etc.

– Identify content and structure of software artefacts

To use LATINO, a selection of the learning resources relevant to the intended ontology (and that were identified in the previous data acquisition stage) should be selected. Given a concrete TAO scenario, the first question to be answered by a software engineer is – what are the *text-mining instances* (i.e. pieces of data to be used for text-mining)[10], in this particular case. The user needs to study the data at hand and decide which data entities will play the role of instances in the transitioning process. It is impossible to answer this question in general - it depends on the available sources. The cookbook offers users some potential choices including Java/C++ classes, methods, and database entities. In the GATE case study, the instances are all Java classes.

Next, we need to assign a textual document (description) to each text-mining instance. This step is not obligatory, as there is no universal standard for which text should be included. However, it is important that only relevant elements of text are included to avoid the text-mining algorithms generating poor or misleading results. Users should develop several (reasonable) rules for what to include and what to leave out, and evaluate each of them in the given setting, choosing the rule that will perform best. Some of the more commonly used rules are given in the cookbook. Given that the GATE Java classes were used as text-mining instances for the GATE case study, the assigned textual documents included the Java classes' *class comment*, *class name*, *field names*, *field comments*, *method names* and *method comments*.

The user may also identify any structural information evident from the data. This step is also not obligatory, provided that textual documents have been attached to the instances. The user should consider any kind of relationships between the instances (e.g. links, references, computed similarities, and so on). Note that it is sometimes necessary to define the instances in a way that makes it possible to exploit the relationships between them. For Java/C++ classes, the potential links that can be extracted include inheritance and interface implementation graphs, type reference graphs, class and operation name similarity graphs, comment reference graphs, etc. After this step, the data pre-processing phase is complete. More information about those types of links and the different calculations of link weight can be found in [11].

– Creating feature vectors from contents and structures

The text-mining algorithms employed by LATINO (and also by many other data-mining tools) work with feature vectors. Therefore, once the text-mining instances have been enriched with the textual documents, they need to be converted into feature vectors. LATINO is able to compute the feature vectors from a document network, based on the source code. In such networks, classes generally contain methods that have

[9] `http://www.tao-project.eu/researchanddevelopment/demosanddownloads/ontology-learning-software.html`

[10] To avoid confusing the term INSTANCE in the text-mining sense with the term INSTANCE from the ontological perspective, we will talk about TEXT-MINING INSTANCES to emphasise the text-mining context.

some return value which may in turn correspond to instances of another class. Comments within the source code may also refer to other classes. For each of these cases, a graph can be created, where vertices represent Java classes and edges represent references between these classes. This may result in the construction of several graphs which all share the same set of vertices. Different weights (ranging from 0 to 1) are assigned to each graph. To help the user set the parameters, the TAO Suite application OntoSight [10] provides the user with an insight into the resulting document networks and semantic spaces via interactive visualisation tools. These feature vectors are further used as an input for OntoGen[11] which is a semi-automatic data-driven ontology construction tool that creates suggestions for new concepts for the ontology automatically. OntoGen is also integrated with LATINO and the TAO Suite.

– Create domain ontology from feature vectors.

The most important step of ontology development is to identify the concepts in a domain. Using OntoGen, this can be performed by using either a fully automated approach such as unsupervised learning (e.g. clustering), or a semi-automated supervised learning (e.g. classification) approach.

The main advantage of unsupervised methods is that they require very little input from the user. The unsupervised methods provide well-balanced suggestions for sub-concepts based on the instances and are also good for exploring the data. The supervised method provided by OntoGen, however, requires more input. The user has to initially identify the desired sub-concept, and then describe it through a query before engaging in a sequence of questions to clarify that query. This is intended for the cases where the user has a clear idea of the desired sub-concept that should be added to the ontology, but where this sub-concept is not automatically discovered by the unsupervised method. For the GATE case study, the unsupervised approach has proven to be sufficient for this learning task, as there is little prior knowledge regarding the desired concepts for inclusion within the ontology. Further details on using OntoGen/LATINO can be found in [11].

It is important to note that the automated methods are not intended to extract the perfect ontology, they only offer support to domain experts in acquiring this knowledge. This help is especially useful in situations such as in the Gate scenario, where the knowledge is distributed across several documents. Therefore, the automatically acquired knowledge is post-edited, using an existing ontology editor, to remove irrelevant concepts and add missed ones. This activity occurs during the *Design Ontology* stage within the TAO cookbook (as illustrated in Figure 1). Hence, ontology learning tools are seen as a support for generating ontologies, and using them makes sense only in the context of large legacy applications (i.e. where "thousands" of documents are used). These tools can offer guidance in these cases, when building ontologies from scratch might be impractical.

After creating the domain ontology, it is saved within the TAO repository for later use. It is only now possible to augment the existing content of a legacy application (including any service definitions) semantically. We present the details in the following subsection.

2.1.3. Service and content augmentation

Content Augmentation is a specific metadata generation task that facilitates new information access methods. It enriches the augmented text with semantic information, linked to a given ontology, thus enabling semantic-based search over the annotated content. For legacy software applications, the key parts are the service descriptions, software code and the documentation. While there has been a significant body of research on semantic annotation of textual content (in the context of knowledge management applications), only limited attention has been paid to processing legacy software artefacts, and in general, to the problem of semantic-based software engineering. As part of the TAO Suite, a tool known as the *Key Concept Identification Tool (KCIT)* was developed to assist users in annotating heterogeneous software artefacts semi-automatically. In essence, *KCIT* is capable of performing two tasks: 1) *semantic annotation*, whereby different elements within a document (such as phrases, n-grams or terms) are identified using various information extraction techniques, and linked to concepts within an ontology; and 2) *document query*, whereby the annotated documents are first stored within a persistent storage repository (i.e. the TAO Repository), and then retrieved using a relevance-query mechanism which exploits a selected set of semantic annotations to find relevant documents, rathe than using keyword-lookup techniques. More information about *KCIT* can be found at [5].

[11] http://ontogen.ijs.si/

To use *KCIT*, we first need to identity which Web services users want to provide, and also which types of related content that need to be annotated.

– Identify services and other content to be annotated.

The first step in creating a Web service is to design and implement the application that represents the Web service. This step includes the design and coding of the service implementation, and the verification of all of its interfaces (to determine whether or not they work correctly). Once the Web service has been developed, the service interface definition can be generated from the implementation of the service (i.e. the service interface can be derived from the application's Application Programming Interface (API)). Web service interfaces are usually described as WSDL documents that define the interface and binding of the corresponding Web service implementations. In this paper, we assume that the Web services and the corresponding WSDL definitions for a legacy application have already been developed. As various methods and tools for wrapping the existing functionalities of a legacy application as services currently exist [16,8], we focus here on assisting users in the annotation of existing WSDL definitions to generate SA-WSDL definitions[12].

– Annotate automatically and manually

KCIT identifies key concepts from software-related legacy content by preprocessing ontology lexicalisations (e.g. replacing dashes and underlines with spaces, splitting camelCased words, etc.) and finally extracting the root from each ontology lexicalisation. It is this root that is matched against the roots of the words in text. Hence, KCIT performs more than an exact text match, like many other existing approaches. It can also be configured to better adopt different use cases. For example, when preparing a document such as WSDL, it can be configured so that the tags' processing is enabled. Users then just click a button and KCIT goes through the WSDL file or other legacy content and automatically identifies the pieces of text or tag, which are related to concepts or relations defined in the domain ontology by using NLP techniques. After the process of automatic annotation is finished, users can validate results by visualising them (by using GATE GUI for example), correcting annotations if necessary, and adding new ones by manually selecting the text they want to link to the relevant concept from the ontology. Figure 3(a) shows an example of annotated GATE WSDL file. We can see details of the highlighted annotation over the $clie$ string, where it shows the instance URI (which refers to $clie$) and the classURI (which refers to Corpus Pipeline). By automatically processing WSDL files using the TAO Suite, we produce the SWS descriptions represented using SA-WSDL. The TAO Suite can also be used to annotate other software artefacts including user guides, developer guides, forum posts, source code, etc. Figure 3(b) shows the results of processing the GATE class $FlexibleGazetteer.java$. The popup 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 (property-Value) of the property rdfs:label (propertyURI) for an instance (type) that is of type GATE developer (classURI). The automatically annotated results could contain some flaws, and we need to ensure that this semantic metadata is correctly asserted. The TAO Suite allows domain experts to manually check the correctness of the annotations.

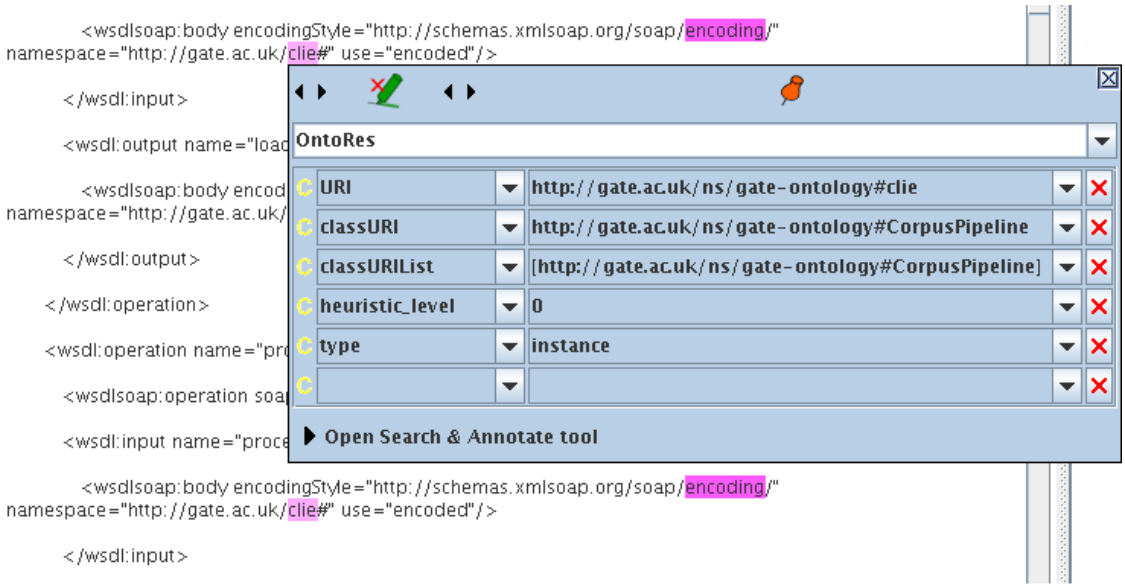
– Storing and Querying Annotations

In order to access the semantic knowledge, the resulting annotation features, together with document-level metadata, are read and exported in a format which can be 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 the documents. For example, if a document contains mentions of the class *Sentence Splitter*, the output should be modelled 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, the PROTON Knowledge Management ontology[13] has been used in our repository, through which the information about the type and address of a document, the position (the start and end offset) of a 'mention' within a document can be represented in a standard way.

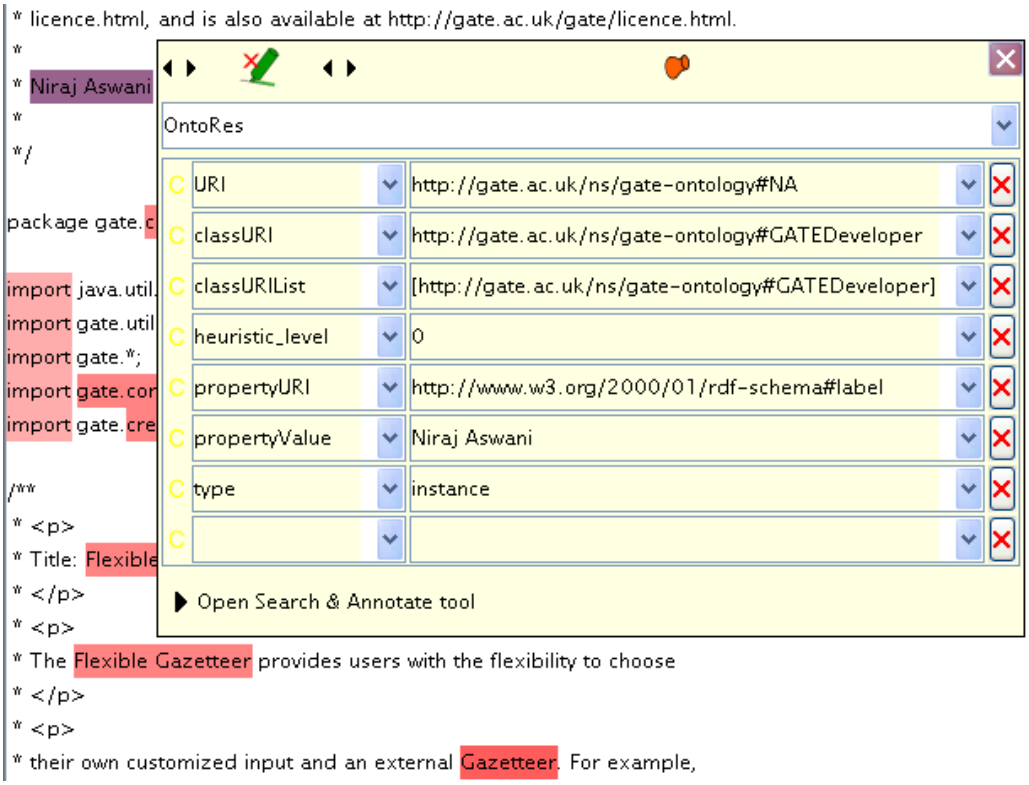
The extracted annotations are stored in our OWL-compatible knowledge repository (OWLIM [14]), and accessible for querying using formal SW query languages (e.g. SPARQL). The exported annotations represented using OWL are stored separately from the actual GATE ontology (used for content augmentation). This way, we can easily keep the annotations and the text syncronized.

[12] Semantic Annotations for WSDL (SA-WSDL) [15] is the latest W3C recommendation for describing Semantic Web Services.

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



(a) Annotate WSDL file



(b) Annotate Java code

Fig. 3. CA interface

Languages for querying OWL such as SPARQL, 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 text-based queries. In order to enable advanced semantic-based access through text-based queries, a *Question-based Interface to Ontologies* – QuestIO has been developed and integrated in the TAO Suite. 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 first recognises key concepts inside the query, detects any potential relations between them, and then creates the required semantic query. An example query with results is shown in Figure 4; for the query 'niraj', a 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 3(b), from which it can be concluded that $Niraj$ is the author of the class $FlexibleGazetteer.java$. The advantage of the semantic query is that queries are observed as concepts, rather than as a set of characters – as is the case in traditional search engines. I.e., $Niraj$, $Niraj$ $Aswani$, or NA (as initials) would all return the same results as soon as the ontology encodes that these terms refer to the same concept.

3. Evaluation and discussion

3.1. Evaluation

Ontology Learning from Software Artefacts. Most ontology evaluation approaches that have been proposed in the literature rely on the opinions and common sense of domain experts. For example, [18] proposed an approach whereby an expert ontology engineer was asked to model a gold standard for the task and compare it with the generated ontology. Another study proposed that domain experts should use the ontology in an application and then evaluate the results [21]. A set of ontology criteria have been designed [17], which need to be assessed manually by domain experts, based on common sense and domain knowledge.

In our evaluation, we have used a combination of these approaches. After the ontology has been learned from the software artefacts, we asked GATE developers to refine it in order to create a gold standard for the comparison. The final ontology with populated instances is available from `http://gate.ac.uk/ns/gate-kb`.

A questionnaire was designed to collect the general opinion from engineers about the learned ontology. The feedback from experts is encouraging and details can be found in the reports [4,23]. To measure the quality of the GATE ontology that is created by the TAO method, we took a sample of 36 questions collected at random from the GATE mailing list and measured what percentage can be answered using the developed ontology. In these questions, numerous GATE users enquired about GATE modules, plugins, processing resources, and problems they encounter while using these components. After examining these questions, we identified that out of these 36 questions, 61.1% (22 questions) were answerable: the GATE domain ontology that was developed following the TAO methodology contained the answers to these questions. The questions were mainly factual questions enquiring about GATE components such as *What are the runtime parameters of the ANNIE POS Tagger?*. The remaining 14 questions (38.9%) were unanswerable: the answer was not in the ontology/knowledge base. Most of unanswerable questions tended to enquire about specific features that were not included in user manuals and documentation, but were only known by experienced GATE developers. In additon, some questions enquired about personal problems without enough explicit information such as *I cannot get Wordnet plugin to work*.

Content Augmentation. To evaluate the CA component, we have selected 20 documents to serve as a representative corpus of GATE software artefacts, including forum posts, java classes, and the user manual. We have first manually annotated these documents to create a gold standard corpus. Next, we ran KCIT to automatically annotate these 20 artefacts, and then we compared the results using precision and recall; we have achieved an average precision of 94.28% precision and a recall of 96.99%. For more information about this experience and the evaluation for other software components such as *Heterogeneous Knowledge Store* and *QuestIO*, please refer to the report [4].

User-centric Evaluation of the Transitioning Results. In order to conduct a user-centric evaluation and investigate benefits of the TAO transitioning tools, we chose to test an integrated testbed containing user-understandable content. In other words, we asked a group of GATE developers and users to carry out a set of tasks involving the source code, software doc-

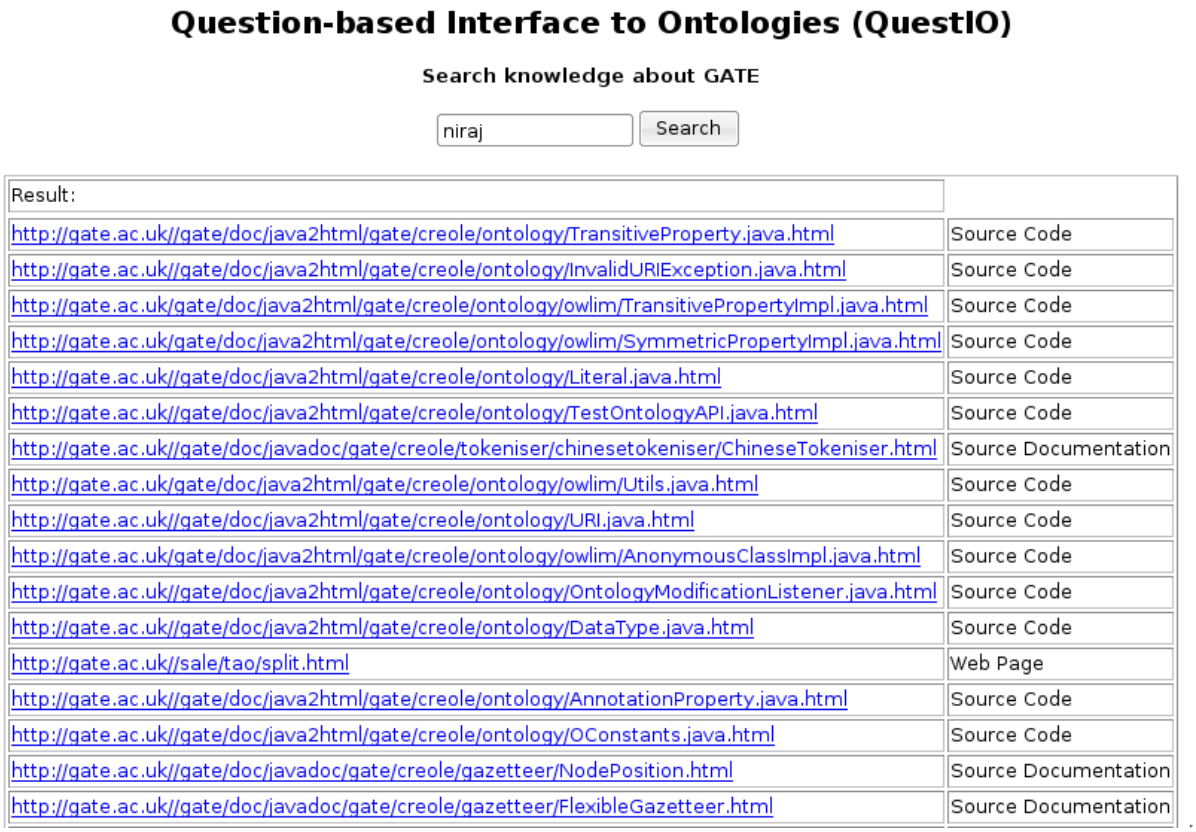


Fig. 4. List of results for the query 'niraj'

umentation, and other human-readable software artefacts (e.g. to 'Find which forum posts are related to the Learning PR'), excluding the semantically annotated services, as the latter are aimed at automatic processes and are hard to work with for non-specialists. However, the same TAO tools can be used for discovering semantically annotated services. In fact, being able to handle these diverse types of legacy services and data with the same toolset was one of the original objectives of the project.

The aim of this qualitative evaluation is to validate whether software developers, who are not experienced in Semantic Web technologies and formalisms, are able to find easily all information relevant to their tasks by using the semantic-based testbed. This new semantic-based system was evaluated with developers and users of the GATE open-source platform, in order to compare their working practices at present and with the new technology. In a nutshell, we carried out a repeated measure: task-based evaluation design (also called within-subjects design), i.e., the same users interact with our prototype (further referred as *new*) and also use their current working practices and tools (further referred as *traditional*), in order to complete a given set of tasks.

From the study, we measured:

- **Efficiency: time spent to complete the tasks using the two approaches – traditional and new.** On average, it took 46.61% longer to finish all the set tasks using the traditional methods in comparison to the new semantic-based prototype (107.1375 seconds $vs.$ 157.075 seconds).
- **Efficiency: the percentage of completed tasks using the two approaches.** Overall, the success rate for performing tasks using the prototype was 152.11% better in comparison to the success rate using the legacy system (0.355 in comparison to 0.895, on a scale from 0 to 2).
- **User satisfaction: the SUS questionnaire as a standard satisfaction measure.** We chose the SUS questionnaire as our principal measure of software usability because it is a de facto standard in this field. SUS scores range from 0 (very little

satisfaction) to 100 (very high satisfaction). Total score in our evaluation was 69.38.

Detailed study results are in the report [4]. With regards to justification of using the TAO tools in comparison to other available ones, we refer the reader to [20].

3.2. Impact and exploitation

The TAO method and tools offer a low-cost migration path for legacy applications to knowledge technologies and is accessible to both SMEs (which are cost sensitive) and large enterprises (with huge investments in complex and critical IS). The results have been validated in two high-profile case studies: a comprehensive open source platform (with thousands of users) and a data-intensive business process application (managing a multi-million business). More information about these case studies is in [5,6].

TAO project partners have obtained over €750,000 in follow-up commercial funding from the Austrian company Matrixware to apply two of the TAO tools to the problem of Large-Scale Semantic Annotation of Patents. The goal is to exploit this TAO technology and annotate terabytes of data in several days of supercomputer time. The TAO Suite is also now used by the company behind `http://videolectures.net` for the automatic classification of video materials posted to their web site.

TAO has delivered a series of tutorials focused around the TAO Suite and also organised an industry-oriented workshop in January 2009 which attracted strong interest and produced very positive feedback on the technological achievements of the project; one company commented that they had been waiting for such enabling technology for their business cases, and another company noted that it was the first time that they had seen complementary solutions (ontology learning, content augmentation, knowledge storage and queries, WSDL annotations, SOA architecture etc.) harnessed together to facilitate the overall process of transitioning. In other words, from this external industrial perspective, TAO has been successful in developing and integrating the necessary enabling technologies for transitioning legacy applications to ontologies, without making too rigid a stance on what software architectures or semantic service formalisms must be adopted.

3.3. License status and the latest development

The TAO Suite is open source and Eclipsed-based and can be freely downloaded from the project's web site[14]. The ontology learning tool LATINO, the content augmentation tool KCIT and the knowledge store HKS are available under LGPL license.

The TAO project partners are currently maintaining and will further develop these software components. LATINO will be developed further by one of the TAO project partners (the Jozef Stefan Institute - JSI)[15], as part of several ongoing EU projects. KCIT and the other related components have been integrated into GATE[16], and are being further developed as part of the GATE development process. The latest version of GATE 6.0 was released in November 2010 and a new release is planned for May 2011. HKS is further developed as part of the OWLIM Semantic Repository (OWLIM)[17] by Ontotext[18].

4. Related work

A number of ontology-design methodologies that have been proposed to date to guide the process of ontology development from scratch have been listed in a comprehensive survey in [13,9]. While [7] has identified seven of the most commonly used methodologies for designing ontologies from scratch, [12,22] have outlined a set of principles and design criteria that have been proved useful in developing domain ontologies. During the last decade several ontology-learning systems have been developed such as ASIUM, OntoLearn, Text2Onto, OntoGen, and others. Most of these systems depend on linguistic analysis and machine learning algorithms to find potentially interesting concepts and relations between them.

Whilst several methodologies exist to develop domain ontologies either from scratch or from text, there is no widely accepted method for transitioning existing applications to SOA based on domain ontologies. [16] proposed the use of black-box wrapping techniques to migrate functionalities of existing Web applications to traditional Web services. In our methodology, the do-

[14] `http://www.tao-project.eu/researchanddevelopment/demosanddownloads/tao-suite.html`
[15] `http://www.ijs.si/ijsw/JSI`
[16] `http://gate.ac.uk/`
[17] `http://www.ontotext.com/owlim/`
[18] `http://www.ontotext.com/`

main ontology plays a key role in the transition process as it contains all the semantics required for annotating the services of the new SOA. Our method and tools are focused on legacy application transitioning. We use various kinds of function related resources to derive the domain ontology. Since most existing applications tend to have documentation describing their functionality and APIs, it is possible to use automatic processing tools to abstract domain concepts from those terms used in such documentation and build the domain ontology. Our methodology is also fully supported by an integrated tool studio.

5. Conclusion and future work

A key requirement of transitioning applications to Semantic Web Services has promoted the urgent need of systematic methodologies and tools to assist the migration process. In this paper we present the TAO methodology and tool suite for transitioning legacy applications to SWS, which allows users to migrate their applications to SWS platform automatically or semi-automatically. In the future, more case studies will be applied to further evaluate the system. We also plan to integrate some third party tools to our framework, such as WSDL generation tool, to make TAO Suite more complete and flexible.

References

[1] Florence Amardeilh, Bernard Vatant, Nicholas Gibbins, Terry R. Payne, and Hai H.Wang. Sws bootstrapping methodology. Technical Report D1.2.2, TAO Project Deliverable, 2009. http://www.tao-project.eu/resources/publicdeliverables/d1-2-2.pdf.

[2] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. MIT Press, Cambridge, MA, 2. edition, 2008.

[3] Kalina Bontcheva, Ian Roberts, Milan Agatonovic, Julien Nioche, and James Sun. Case study 1: Requirement analysis and application of tao methodology in data intensive applications. Technical Report D6.1, TAO Project Deliverable, 2007. http://www.gate.ac.uk/projects/tao/webpage/deliverables/d6-1.pdf.

[4] Damljanovic D. and Bontcheva K. Gate case study: Experiments and evaluation of testbed. Technical Report D6.4, TAO project, 2009. http://www.tao-project.eu/resources/publicdeliverables/d6-4.pdf.

[5] Damljanovic D., Bontcheva K., Tablan V., Roberts I., Agatonovic M., Andrey S., and Sun J. Gate case study: Domain ontology and semantic augmentation of legacy content. Technical Report D6.2, TAO project, 2008. http://www.tao-project.eu/resources/publicdeliverables/d6-2.pdf.

[6] Cerbah F. Case study 2: Domain ontology building and semantic augmentation of legacy content. Technical Report D7.2, TAO project, 2008. http://www.tao-project.eu/resources/publicdeliverables/d7-2.pdf.

[7] Mariano Fernandez-Lopez, Asun Gomez-Perez, Jerome Euzenat, Aldo Gangemi, Y. Kalfoglou, D. Pisanelli, M. Schorlemmer, G. Steve, Ljilajana Stojanovic, Gerd Stumme, and York Sure. A survey on methodologies for developing, maintaining, integration, evaluation and reengineering ontologies. Ontoweb deliverable, Universidad Politecnia de Madrid, 2002. `http://www.aifb.uni-karsruhe.de/WBS/ysu/publications/OntoWeb_Del_1-4.pdf`.

[8] Gerald C. Gannod, Huimin Zhu, and Sudhakiran V. Mudiam. On-the-fly wrapping of web services to support dynamic integration. In *WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering*, page 175, Washington, DC, USA, 2003. IEEE Computer Society.

[9] Asunción Gómez-Pérez and David Manzano-Macho. An overview of methods and tools for ontology learning from texts. *Knowl. Eng. Rev.*, 19(3):187–212, 2004.

[10] Miha Grcar. Ontology learning services library. Technical Report D2.2.2, TAO Project Deliverable, 2008. http://www.gate.ac.uk/projects/tao/webpage/deliverables/d2-2-2.pdf.

[11] Miha Grcar, Dunja Mladenic, Marko Grobelnik, Blaz Fortuna, and Janez Brank. Ontology learning implementation. Technical Report D2.2, TAO Project Deliverable, 2007. http://www.gate.ac.uk/projects/tao/webpage/deliverables/d2-2.pdf.

[12] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[13] Dean Jones, Trevor Bench-Capon, and Pepijn Visser. Methodologies for ontology development. In *Proceedings of IT&KNOWS Conference of the 15 th IFIP World Computer Congress*, pages 62–75. Chapman and Hall Ltd, 1998.

[14] Atanas Kiryakov, Damyan Ognyanov, and Dimitar Manov. Owlim – a pragmatic semantic repository for owl. In *Int. Workshop on Scalable Semantic Web Knowledge Base Systems*, pages 182–192, New York City, USA, 2005. Springer-Verlag.

[15] Holger Lausen and Joel Farrell. Semantic annotations for WSDL and XML schema. W3C recommendation, W3C, August 2007.

[16] Giusy Di Lorenzo, Anna Rita Fasolino, Lorenzo Melcarne, Porfirio Tramontana, and Valeria Vittorini. Turning web applications into web services by wrapping techniques. In *WCRE '07: Proceedings of the 14th Working Conference on Reverse Engineering*, pages 199–208, Washington, DC, USA, 2007. IEEE Computer Society.

[17] A. Lozano-Tello and A. Gómez-Pérez. ONTOMETRIC: A Method to Choose the Appropriate Ontology. *Journal of Database Management*, 15(2), April-June 2004.

[18] Alexander Maedche and Steffen Staab. Measuring similarity between ontologies. In *EKAW '02: Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web*, pages 251–263, London, UK, 2002. Springer-Verlag.

[19] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[20] Tomás Pariente, Germán Herrero, Danica Damljanovic, and Farid Cerbah. D1.3.2 case study reports on evaluating the methodology. Technical Report D1.3.2, TAO project, 2009.

http://www.tao-project.eu/resources/publicdeliverables/d1-3-2-final.pdf.

[21] Robert Porzel and Rainer Malaka. A task-based approach for ontology evaluation. In *Proceedings of ECAI 2004 Workshop on Ontology Learning and Population*, Valencia, Spain, 2004. Springer-Verlag.

[22] Bill Swartout, Ramesh Patil, Kevin Knight, and Tom Russ. Toward distributed use of large-scale ontologies. In *10th Workshop on Knowledge Acquisition*, Canada, June 1996.

[23] Valentin Tablan, Danica Damljanovic, and Kalina Bontcheva. A natural language query interface to structured information. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 361–375, Berlin, Heidelberg, 2008. Springer-Verlag.

[24] Marinova Z. Heterogeneous knowledge store. Technical Report D4.2, TAO Project Deliverable, 2008. http://www.gate.ac.uk/projects/tao/webpage/deliverables/d4-2.pdf.