Transition of Legacy Systems to Semantic Enabled Application: TAO Method and Tools

Hai H. Wang a,∗, Danica Damljanovic d, Terry Payne b, Nick Gibbins c, and Kalina Bontcheva d

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 very tedious and expensive process, which carries a definite 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 allows users to migrate their applications to Semantic Web Services platform automatically or semi-automatically. The transition of GATE system is used as a case study.

Keywords: Semantic Web Services, Annotation

1. Introduction

Semantic Web (SW) and Semantic Web Service (SWS) [18] have been recognised as very promising technologies to emerge, exhibiting huge commercial potential and attracting significant attention from both industry and the research community. Despite the great prospect of success, the industrial take-up of SW and SWS technologies have 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 semantically web-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 an urgent need to therefore provide strategies that support the construction of ontologies which facility the migration of legacy systems to Semantic Web Services platforms, and also tools to support such strategies.

This paper reports 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

∗Corresponding author. E-mail: H.WANG10@aston.ac.uk
Applications to Ontologies (TAO) project\(^1\). In TAO, which is in the European Sixth Framework Program, we create 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., TAO Suite software, 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 cookbook-style guideline on 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 about the important phases to be performed to the transition process of legacy systems to semantic enabled application following the TAO scenario and it is fully supported by TAO Suite, which is integrated from several software components. Figure 2 shows the architecture of the transitioning environment. In it, the ontology learning tool is used to derive domain ontology from legacy application documents (specifications, UML diagrams, code documents, software manuals, incl. images). The content augment 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 contents, domain ontology and semantic annotations. An Integrated Development Environment (IDE) is 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 this to emphasis the contribution made by the description of an existing body of legacy application (e.g., application APIs and developer documentations, SOA design documentations, and so on). We call this refinement Service-Oriented Ontology Learning.

-- Using domain ontologies to augment semantic content and service. The service annotation process described in many existing SOA design methodologies refers to the description of services at the signature level in languages like 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 datatypes based on XML Schema, rather than richer knowledge-based types taken from an ontological characterisation of the 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 ontology.

In the next subsection, we present a cook-book style guideline about the usage of TAO tools. Note that this paper only focuses on the usage of TAO tools, due to the limited space. For more technical details about various tool components, please refer to the respected reports, which can be downloaded at http://www.tao-project.eu/.

To better illustrate the idea, we use the transition of GATE\(^2\) system to semantic enhanced services as a case study. 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 the large amount of heterogeneous information that cannot be accessed via a unified interface.

The advantages of transitioning GATE to semantic enhanced services are two-fold. Firstly, GATE components and services will be easier to be discovered and

\(^1\)http://www.tao-project.eu/

\(^2\)http://gate.ac.uk
integrated within other applications due to the use of Semantic Web service technology. Secondly, users will be able to find easily all information relevant to a given GATE concept, searching across all different software artefacts: the GATE documentations, 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. [2] presents the detailed view of the advantages and possibilities arising from building domain ontology and application of semantic enrichment of software artefacts in the GATE case study.

2.1. Transitioning cookbook

TAO methodology has three main phases: the knowledge 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 to illustrate the main transitioning process and we give more detailed explanation of the major activities in the following.

To transition a given legacy application into semantic enabled services, the software engineers first check if there are some previously developed ontologies for the application. Some public ontology search engines...
or public ontology libraries can be used for this task\(^3\). 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 NEON methodology\(^4\), directly adopt and extend the found ontology. However, according to our past ontology developing experiences, these approaches are not ideal for many use cases. The main reason is that for most of the time, it is difficult to find a perfectly matched ontology for a legacy application. Re-using complex domain ontologies built for the domain in similar projects could be a tedious task. The more complex an ontology is, and the more tied it is to its original context of development and use, the less likely it is to fit another context. It’s often as difficult and costly to trim such ontologies in order to keep only the relevant parts than to re-build those parts completely. Building domain ontologies with a “top-down” approach as extensions of “foundational ontology” is another popular approach. Foundational ontologies are often highly abstract and constraining. Besides, they are almost never adapted to business requirements. They bear strong constraints that are rarely part of the requirements for the system to build. In TAO method, if an related ontology is found (either directly existed or obtained on transformation of knowledge aware resources, such as thesauri, lexicons, database schemas), it is saved into the knowledge store developed by TAO and used as an important training data for the ontology learning tool together with other software artifacts. For the GATE case, we develop the ontology from scratch with the assistance of TAO Suite.

2.1.1. Knowledge acquisition

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

-- Resources collection

In the TAO cookbook, we have identified some data sources which are commonly relevant to the description of a legacy application, such as application source code, APIs, JavaDoc etc. For more information about the potential data sources which may be related to the description of a legacy systems and their classification, please refer to the report [1]. For the GATE case study, first Java source code and JavaDoc files are collected. Those documents can be downloaded from \texttt{http://gate.ac.uk/download/index.html}.

-- Save the resource corpuses to TAO Repository

After collecting all the related data sources, we store them in the TAO repository [22] – a component of TAO Suite. It 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.

2.1.2. Ontology Learning

The purpose of ontology learning from software artefacts is essentially discovering concepts and relations from 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. Previously, we have presented a detailed review of different ontology learning approaches [1]. [8] also reviewed the major methods for semi-automatically building ontologies from texts. LATINO\(^3\), a component of TAO Suite, is used for Ontology Learning. LATINO is a more-or-less general data-mining framework that joins text mining and link analysis for the purpose of (semi-automated) ontology construction. LATINO has at least two novelties comparing with some existing ontology learning methods. Firstly, using LATINO, the ontologies are constructed from the knowledge extracted from the data that accompany typical legacy applications. A set of important data sources related to the functionalities of a legacy application and their inter/intro-relationships are carefully studied and used in the ontology construction process. We introduce the term “application mining” which denotes the process of extracting this knowledge. Secondly, LATINO is not only limited to texts data sources. Additional data resources that can be used for ontology learning include structure documents, such as database schema, UML models, existing source code, API, etc., or textual documents, such as requirement documents, manuals, forum discussions etc.

-- Identify content and structure of software artifacts

\(*\) Identify the text-mining instances

\(*\) Assign textual document to instances

\(^3\)\texttt{http://swoogle.umbc.edu} or \texttt{http://swse.deri.org}

\(^4\)\texttt{http://www.neon-project.org/}

\(^5\)\texttt{http://www.tao-project.eu/researchanddevelopment/demosanddownloads/ontology-learning-software.html}
To use LATINO, we need to first identify the contents and structures of those related resources collected in the previous step. Given a concrete TAO scenario, the first question to be answered by a software engineer is – what are the text-mining instances? (which are used as graph vertices when dealing with the structure) in this particular case, i.e., the user need 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, database entities and etc. In the GATE case study, the instances are Java classes.

Next, we need to assign a textual document (description) to each text-mining instance. This step is not obligatory. There is not a universal standard for which text should be included, but it is important to include only those bits of text that are relevant and will not mislead the text-mining algorithms. 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 common used rules are given in the cookbook, e.g., for most legacy applications that have well-commented Java/C++ source code available, class comment, class name, field names, field comments, method names and method comments can be used.

The user may also identify the structural information, which is 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 graph, type reference graph, class, operation name similarity graph, and comment reference graph, 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 the report [10].

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, we need to convert them into feature vectors. LATINO is able to compute the feature vectors from a document network. When this network is created based on the source code, such as in the GATE case study, it is common that a class has methods that return values of the type represented by another class. Also, comments in Java classes usually refer to other classes. For each of these cases, one graph would be created. In these graphs, vertices represent Java classes and edges represent references between these classes. After creating several such graphs they all have the same set of vertices. Next, different weights (ranging from 0 to 1) are assigned to each graph. To help the user set the parameters, TAO Suite includes an application OntoSight [9], which gives the user insight into document networks and semantic spaces through visualization and interaction, has been developed. For the usage of LATINO, we refer reader to [9].

These feature vectors are further used as an input for OntoGen which is a semi-automatic data-driven ontology construction tool that creates suggestions for new concepts for the ontology automatically. OntoGen is integrated with LATINO and TAO Suite.

Create domain ontology from feature vectors.

The most important step of ontology development is identifying 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 on the other hand requires more input. The user has first to figure out what the sub-concept is, then to describe the sub-concept through a query and go through the sequence of questions to clarify the query. This is intended for the cases where the user has a clear idea of the sub-concept he wants to add to the ontology, although the unsupervised methods is not capable to discover it. For the GATE case study, we have chosen the unsupervised approach, because we have little knowledge about the ontology. An example of automatically generated concepts visualised using OntoGen is shown on Figure 3. This figure depicts three concepts, namely Nominal Coreferencer, Pronominal Coreferencer and

---

6http://ontogen.ijs.si/
SearchPR. Each of these three concepts represent a separate Processing Resource (PR) in GATE. In OntoGen they are being clustered as belonging to the same group of concepts. If we decide to add this group to the ontology, one class will be created and three instances for each mentioned PR. For more detailed instructions about the usage of OntoGen/LATINO, please refer to the report [10].

![Fig. 3. Concepts derived from GATE source code using OntoGen.](image)

An important point to make is 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 like ours when the knowledge is distributed in several documents. Therefore, the automatically acquired knowledge is post-edited, using an existing ontology editor, to remove irrelevant concepts and add missed ones. We refer this activity as design ontology in our cookbook.

After creating the domain ontology, we can save it into the TAO repository. Now we are ready to augment the existing content of a legacy application (including the service definition) semantically. We present the details in the following subsection.

2.1.3. Service and content augmentation

Content augmentation is a specific metadata generation task aiming to enable new information access methods. It enriches the text with semantic information, linked to a given ontology, thus enabling semantic-based search over the annotated content. In the case of legacy software applications, important parts are the service description, the software code and 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 TAO Suite, TAO has developed a tool named Key Concept Identification Tool (KCIT) to assist users to annotate heterogeneous software artefacts automatically (semi-automatically). In essence, KCIT is capable of performing two tasks: semantic annotation – using Information Extraction some parts of the document content are marked and then linked to an ontology; and, persistent storage and lookup of augmented content, where document retrieval is based on relevance to a selected set of semantic annotations instead of relevance to words (like in keyword lookup). More information about KCIT can be found at [4].

To use KCIT, we first need to identify which Web services users want to provide and also what kinds of other content need to be annotated.

Identify services and other content to be annotated.

Note that normally 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 testing to verify that all of its interfaces work correctly. After the Web service is 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 services interfaces are usually developed in 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. There are some existing methods and tools for wrapping the existing functionalities of a legacy application as services [15,7]. Here we focus on helping users to annotate the existing WSDL definitions to get SA-WSDL definitions. SA-WSDL [14] is one of the latest W3C recommendation for Semantic Web Service.

Annotate – automatically and manually

KCIT identifies key concepts from software-related legacy content intelligently (more than exact text match, like many other existing approaches). It can also be configured to better adopt different use cases. E.g., 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 e.g. in GATE GUI, correcting annotations if necessary, and adding new ones by manually selecting the text they want to link to the relevant concept from the ontology. E.g., Figure 4(a) shows an example of annotated GATE WSDL file. We can see details of the highlighted annotation over the `client` string, where it shows the instance URI (which refers to `client`) and the classURI (which refers to Corpus Pipeline). By automatically processing WSDL files using TAO Suite, we produce the SWS descriptions – SA-WSDL files. TAO Suite can also be used to annotate other software artefacts including user guides, developer guides, forum posts, source codes and etc. Figure 4(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 automatic annotating results could contain some flaws, and we need to ensure that these semantic metadata are correctly asserted. TAO Suite allows domain experts to manually check the correctness of them.

### Storing and Querying Annotations

In order to access the semantic knowledge, the produced 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. E.g., 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\(^7\) 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.

\(^7\)http://proton.semanticweb.org/2005/04/protonkm

The extracted annotations are stored in our OWL-compatible knowledge repository (OWLIM [13]), and accessible for querying using formal SW query languages (e.g. SQARQL). 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. 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 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 works so that it first recognises key concepts inside the query, detects any potential relations between them, and creates the required semantic query. An example query with results is shown in Figure 5. 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 4(b), from which it can be concluded that Niraj is the author of the class FlexibleGazetteer.java. The advantage of the semantic query is such that queries are observed as concepts, not like a set of characters – as it is the case in traditional search engines. E.g., 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.

Fig. 5. List of results for the query ‘niraj’

3. Evaluation and discussion

3.1. Evaluation

Ontology Learning from Software Artefacts. Most of ontology evaluation approaches proposed in the literature rely on domain experts’ options and common sense. E.g., [17] proposed to ask an expert ontology engineer to model a gold standard for the task and compare it with the generated ontology. [19] suggested to let domain experts use the ontology in an application and then evaluate the results. [16] defines a set of ontology criteria, 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 corpus. Next, we ran KCIT to automatically annotate these 20 artefacts, and then we compared the results using precision and recall. On average 94.28% precision and 96.99% recall rate have been achieved. 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 [3].

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 documentation, 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. 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, which is created using TAO Suite. 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 measures, task-based evaluation design (also called within-subjects using the developed ontology. 36 questions were collected at random from the GATE mailing list, where numerous GATE users enquire 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 is developed following the TAO methodology, contained the answers to these questions. For the remaining 14 questions (38.9%), 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.

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, manual and etc. 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. On average 94.28% precision and 96.99% recall rate have been achieved. 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 [3].
design), i.e., the same users interact with our prototype and also use their current working practices and tools, 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 – old and new**

On average, it took 46.61% longer to finish all the set tasks using the legacy system in comparison to the 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: 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 [3].

### 3.2. Impact and exploitation

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 [4,5].

TAO project partners has obtained over 750,000 euros in follow-on 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 are 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 TAO Suite and also organised an industry-oriented workshop in January 2009 which attracted a strong interest and produced very positive feedback on the technological achievements of the project, one comment being that they were waiting for such enabling technology in their business cases and another being that it was the first time they were seeing 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 a too rigid stance on what software architectures or semantic service formalisms must be adopted.

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 [12,8]. While, [6] has identified seven of the most commonly used methodologies for designing ontologies from scratch. [11,20] 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. E.g. [15] proposed to use black-box wrapping techniques to migrate functionalities of existing Web applications to traditional Web services. In our methodology, the domain 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 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 works

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 TAO methodology and tool 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


