TAO Methodology: Transition of Legacy Systems to Semantic Enabled Application

{ }@cs.man.ac.uk

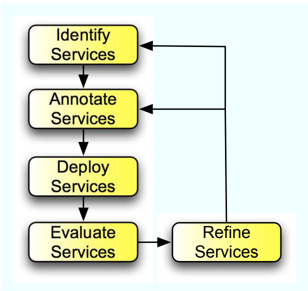
Abstract. Despite expectations being high, the industrial take-up of Semantic Web technologies in developing services and applications has been slower than expected. One of the main reasons is that many systems have been developed without considering the potential of the web in integrating services and 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 which allow the migration of legacy systems to Semantic Web Services platforms, and also tools to support such a strategy. In this paper we propose a methodology for transitioning these applications to Semantic Web Services by taking the advantage of rigorous mathematical methods. Our methodology allows users to migrate their applications to Semantic Web Services platform automatically or semi-automatically.

1 Introduction

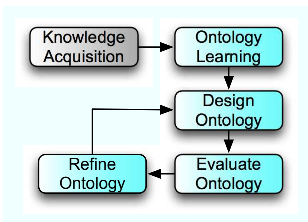
Semantic Web Services combines the Web Services and Semantic Web enabling technologies. By semantically annotating the relevant aspects of declarative Web Service descriptions in a machine-readable format that can facilitate logical reasoning, such service descriptions become interpretable based on their meanings, rather than simply on a symbolic representation. The advantage of this is that many of the tasks involved in using Web Services can be (semi-) automated, for example: discovery, selection, composition, mediation, execution, monitoring, etc. Thus, Semantic Web Service Research [10] has been recognized as one of the most promising technologies to emerge, exhibiting huge commercial potential, and attracting significant attention from both industry and the research community. Despite its great prospect of success, the industrial take-up of Semantic Web Services technologies has been slower than expected. This was 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 rigor 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 Web Accessibility dealing with the transformation of components of the legacy system that are exposed as Web services, Service Transformation where the exposed Web services are mapped to the corresponding Semantic Web Service representations and Semantic Annotation where the Semantic Web Service is annotated using the relevant domain ontology. Without a systematic methodology and proper tool support, the migration from legacy systems to semanticly 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 a strategy.

This paper proposes a new methodology for addressing the above issues in particular Web Accessibility and Componentization, which in turn could lead to an automatic Platform Transformation. The main idea of the methodology is to identify the components/steps for creating web services to represent systems functionality and semantically annotate such services through domain ontologies elicited from system documentations. Typically, the step for creating web services will be merged with the procedures of learning ontologies (from system documentation) to facilitate future ontology pruning and refinement of web service descriptions which eventually leads to bridging the gab of interoperability and hence moving the system closer to SOAs. This methodology is part of Transitioning Applications to Ontologies (TAO) project[1]. (TAO) is a project in the European Sixth Framework Program. The goal of the TAO project is to define methods and tools for transition of legacy information systems to semantic enabled services, enabling semantic interoperability between heterogeneous data resources and distributed applications.

The remainder of this paper is organized as follows. Section 2 provides a high level explanation about the methodology. Section 3 presents cookbook-style guidelines on how to adopt the methodology using the tools developed by TAO. Section 4 discusses some related works. Finally, Section 5 presents the conclusions of this paper and future work.



(a) SOA design lifecycle



(b) Ontology design lifecycle

Fig. 1. SOA and ontology design lifecycle

2 High-level methodology

The transitioning methodology outlined in this section is presented as a high-level composite lifecycle, which highlights the interactions between existing methodologies for developing Service-Oriented Architectures (which include Web Services and Enterprise

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

Architectures), and for ontology design. In this paper, we develop this abstracted lifecycle sketches of SOAs and building domain ontologies from text, and demonstrate how and where these should be linked. The design lifecycle for a service-oriented system is largely divorced from both the specific methodology used to create the system, and from the lifecycle of the individual services within such a system. Figure 1(a) shows a sketch of the design lifecycle for an SOA system.

Service Identification: This process refers both to the identification of existing services which can be repackaged within an SOA system, and also to the identification of required functionality (from a business process modelling exercise, for example) that does not currently exist in operational form, and the subsequent implementation of such functionality as services.

Service Annotation: In order to facility loose coupling of component services through brokerage and matchmaking, it is necessary to describe the services in an SOA system in sufficient detail that a service requester can find an appropriate service that meets their needs.

Service Deployment: Here we refer to the deployment of services within a service execution environment. This may include publishing/advertising services through public service registries, discovering services where by clients can identify candidate services that may fulfil their requirements, selecting services where they chose the most appropriate service(s), composing services where clients integrate several independent services to achieve an overall goal, and finally executing services either through direct invocation or through workflow management systems.

Service Evaluation: This process refers to the ongoing monitoring of an SOA system to determine whether it meets its design goals.

Service Refinement: The refinement of an SOA system typically takes one of three forms: the introduction of new functionalities through the creation of new services; the refactorisation of existing service functionality (through aggregation or further decomposition, for example); or the refinement of the service descriptions to better facilitate service matchmaking and brokerage.

As the design lifecycle for an SOA system is largely independent of the particular methodology used, so the design lifecycle for a domain ontology (Figure 1(b)) can be separated from the specific knowledge acquisition and modelling methodology used. Domain ontologies usually describe the conceptualization of entities, their relationships to each other, instantiations and the axioms related to a specific domain (such as wines or cars). These domain ontologies can either specialise concepts introduced in some other top-level ontologies (which describe very general concepts like space, time, event, which are independent of a particular problem or domain), such as $Dolce$[2] and $OpenCyc$ [3] or they can be created from scratch for a particular domain. A common process to create domain ontologies from scratch can contain several steps:

Ontology learning: *Ontology learning* refers to the use of techniques for automatically or semi-automatically extracting ontologies from existing document corpuses. As

[2] `http://www.loa-cnr.it/DOLCE.html`
[3] `http://www.opencyc.org`

such, the output from an ontology learning process should not be considered as the finished product, but as a first cut that is solidly grounded in the available documentation, and which will inform the later design of a more polished ontology for production use.

Ontology Design: The ontology design process refers to the process of formally codifying the knowledge that has either been manually acquired from a domain expert, or (semi-)automatically extracted from a document corpus. This process may also encompass the identification and reuse of appropriate components within pre-existing ontologies, the alignments of the designed ontology with pre-existing ontologies, or the modularisation of the ontology to facilitate such alignment in future.

Ontology Evaluation: The ontology evaluation process assesses whether or not the designed ontology is fit for purpose.

Ontology Refinement: This process refers to the refactorisation of the designed ontology to better represent the problem domain.

The TAO transitioning methodology provides a logical approach for connecting the above lifecycles (i.e. SOA and Ontology design) through the following three main points.

Learning ontologies from service descriptions: In the 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 have refined this to reflect the contribution made by the structured (but not ontologically-informed) description of an existing body of services (for example, service APIs and developer documentation, SOA design documentation, and so on). We call this refinement *Service-Oriented Ontology Learning*. It is our expectation that the ontology resulting from an automated ontology learning process should be treated as a candidate ontology which will be subsequently be evaluated and refined in the Ontology Design process. Whilst the ontology extraction process may yield some conceptualization of the relevant domain, much of the implicit domain knowledge inherent in a service description will not be captured; thus the extraction of an ontology from structured sources such as those mentioned above may not obviate the need for further work on the domain ontology. However, such structured sources relating to existing services do provide sufficient initial information that suggests the creation of a domain ontology, which can be further evolved.

Using domain ontologies to augment semantic content and service: The Service Annotation process described in the SOA methodology refers to the description of services at the signature level in languages like WSDL[4]. 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.

[4] The signature within WSMO [] describes a functional or atomic interface with inputs and outputs (defined as messanges).

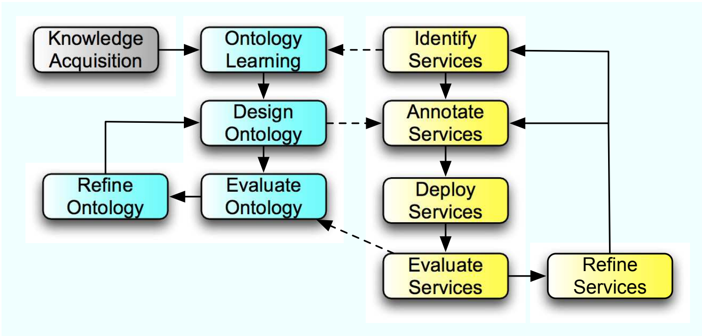


Fig. 2. High-level methodology.

Using feedback from service evaluation to refine ontologies: Both the ontology design lifecycle and the SOA lifecycle contain evaluate-and-refine steps that represent a reflection on the performance of a system and the subsequent reengineering.

Figure 2 depicts an overview of the methodology and a representation of these steps. The interactions between the components of the two life cycles are effectively a refinement of processes within those lifecycles, and reflect the relationship between the products of each individual lifecycle. In particular, we note the task-oriented nature of a domain ontology which is defined with service annotation in mind when compared with a general-purpose ontology for the same domain.

3 Methodology cookbook, tool support and case study

The methodology presented in the previous section provides an abstract view about the important phases needed to be performed during the transition process. In order to support this methodology, TAO project has developed an open source infrastructure and a series of tools to aid the transitioning process. In this section, we present a cook-book style guideline about the usage of TAO tools. To better illustrate the idea, we use the transition of GATE system as a case study.

3.1 GATE as a legacy application – an overview

GATE[5] 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. 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 specic problems that large software architectures encounter and has been chosen as the data-intensive case study in the TAO project.

[5] `http://gate.ac.uk`

The advantage of transitioning GATE to ontologies will be 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, users will be able to use knowledge access tools and nd easily all information relevant to a given GATE concept, searching across all different sotware artefacts: the GATE documentation, XML conguration les, video tutorials, screen shots, user discussion forum, etc. [5] presents the detailed usages of the Semantic enable GATE.

3.2 Transitioning cookbook

As mentioned before, TAO methodology, has three main phases: the knowledge acquisition phase, the ontology learning phase and semantic content and the service augmentation phase. Each phase contains a set of tasks which may interact with each other. Figure 5 presents a UML diagram to illustrate the main transitioning process and we explain major activities in detail.

Given a legacy application, the domain 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[6][7][8]. If such an ontology is found, it can be saved into the knowledge store developed by TAO for future usage, otherwise users have to derive the domain ontology from the legacy software. For the GATE case, we develop the ontology from scratch with the assitance of TAO tools.

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

– ***Resources collection***

We identify some data sources which are commonly relevant to the description of a legacy application, such as application source codes, API, Java Doc etc. For more information about the potential data sources which may related to the description of a legacy systems and their classification, please refer to [2]. In our GATE case, its Java source codes, JavaDoc files are collected. Those documents can be downloaded from `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 repository. TAO project develops a heterogeneous knowledge store to store these data sources. The heterogeneous knowledge store is designed 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. More information about this heterogeneous knowledge store can be found from [11].

[6] `http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library`
[7] `http://swoogle.umbc.edu`
[8] `http://swse.deri.org`

Ontology Learning The purpose of ontology learning from pieces of software is essentially discovering concepts and relations in 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 [2]. In this paper, we show how to learn domain ontologies based on the TAO scenario; LATINO[9], a part of the TAO Suite, supports this. The development of 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. The ontologies are constructed from the knowledge extracted from the data that accompany typical legacy applications. We introduce the term "application mining" which denotes the process of extracting this knowledge.

In the previous step, we collected a set of related data resources that describe the legacy application. To use LATINO to get ontologies from these resources, we need to first identify these resources' contents and structures.

- ***Identify content and structure of software artifacts***
  - ***Identify the text-mining instances***
  - ***Assign textual document to instances***
  - ***Determine the structure of between instances***

Given a concrete TAO scenario, the first question that needs 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. Some potential choices include Java/C++ classes, methods, Database entities and etc. In the GATE case study, the instances are the source code Java classes.

Next, we need to assign as textual document (description) to each text-mining instance. This step is not obligatory, and perhaps not even possible when the data is such that it does not contain any unstructured textual data. Again 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. In general, 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

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

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 [7].

– ***Creating feature vectors from contents and structures***

The text-mining algorithms employed by LATINO (and also many other data-mining tools) work with feature vectors. Therefore, once the text-mining instances have been enriched with the textual documents and discovered structure information, we need to convert them into feature vectors. LATINO is able to compute the feature vectors from a document network. For source code resources (such as the GATE case), it is common that a class has methods that return values of the type rep- resented 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. In extreme case, 0 would be used to exclude the graph, and 1 to include it. Assigning weights is not a trivial process and requires lots of experimenting, experience, and intuition. Following the intuition, the user has to specify the weight setting and examine the results. If the results are not satisfying, the user has to change the settings and repeat the process again. To help the user set the parameters, OntoSight [8], an application that gives the user insight into document networks and semantic spaces through visualization and interaction, has been developed. For the usage of LATINO, please refer to [8]. The feature vectors for the GATE case study can be found at . . . .

These feature vectors are further used as an input for OntoGen[10] which is a semi-automatic data-driven ontology construction tool that creates suggestions for new concepts for the ontology automatically. OntoGen will be integrated with LATINO in later version.

– ***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.

In the unsupervised approach, the system provides suggestions for possible sub-concepts of the selected concept and The supervised approach is based on Support vector machines (SVM) active learning method, which are a set of related supervised learning methods used for classification and regression. The user can start this method by submitting a query. After the user enters a query, the active learning system starts asking questions and labelling the instances. On each step the system asks if a particular instance belongs to the concept. The main advantage of unsupervised methods is that

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

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 to first figure out what should the sub-concept be, he has to describe the sub-concept trough 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 but the unsupervised methods do not 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 gure depicts three concepts, namely *Nominal Coreferencer*, *Pronominal Coreferencer* and *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.

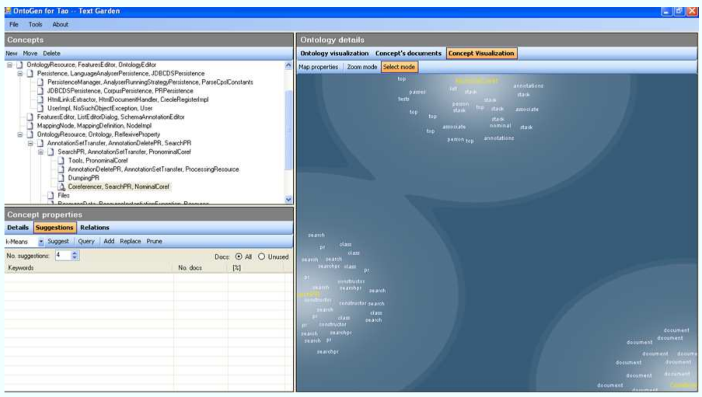


Fig. 3. Concepts derived from GATE source code using OntoGen.

Apart from concept identification, OntoGen/LATINO also implicitly infers subsumption relations between concepts (newer version will also be able to discover some other types of relations).The user can fully customize each of the concepts by defining its instances. The system helps here by detecting outliers both inside and outside the concept. If new data becomes available after the ontology is constructed, the system can help by automatically classifying new instances into appropriate concepts.

For more detailed instructions about the usage of OntoGen/LATINO, please refer to [7].

– ***Design Ontology***

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. In fact no existing OL technique is completely unsupervised: a domain expert must be included somewhere in the knowledge acquisition loop. Therefore, the automatically acquired knowledge is post-edited, using an existing ontology editor, to remove irrelevant concepts and add missed ones. The changes mainly included deleting suggested concepts, as for the ontology it was not important to include too much details in certain cases, for instance distinguishing between the more than 30 types of Exceptions that could be thrown from the different Java classes in the GATE case study.

Furthermore, the resulting ontology should be consistent at different levels. Firstly, the ontology languages have predefined syntax, e.g. RDF/XML syntax. Knowledge represented in these languages must be well formed. Most ontology editors, including LATINO, can be used to check that the ontology is well-formed. Secondly, to meet different usages, ontology languages often comes in various sub-languages or "species". OWL has three different flavours "OWL FULL", "OWL DL" and 'OWL LITE". Thus, the ontology must be built to fall inside the desired species level. For most cases, the user wants to keep their ontologies within the scope of "OWL DL" or "OWL LITE" for ease of reasoning. Tools like the OWL Ontology Validator can be used to check the species of ontology. Furthermore, an ontology cannot contain contradictory information. Therefore, next user needs to make sure that the domain ontology is logically consistent. For example it would be a mistake if we asserted that a pizza was both "Meaty Pizza" and "Vegetarian Pizza" in a knowledge base, given "Meaty Pizza" and "Vegetarian Pizza" are disjoint. Reasoners like Pellet, FaCT++ normally can pick up the logical inconsistency. If an ontology is logically consistent, it does not necessarily follow that it accurately represents the real world. For example, without asserting that "Meaty Pizza" and "Vegetarian Pizza" are disjoint, the ontology is logically consistent even if we define a meaty-vegetarian pizza, even though this is an obvious error. To discover this kind of problems, the ontology needs to be tested by domain experts.

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 subsections.

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 signicant 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. TAO has developed a tool named *Content Augmentation (CA)* to assist users to annotate heterogeneous software artifacts automatically (semi-automatically). In essence, *CA* is capable of performing two tasks: semantic annotation using IE 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 *CA* can be found at [**?**] and it can be accessed at `http://gate.ac.uk/ca-service/services/CAService`.

To use *CA*, we first need to identity which Web services users want to provide and also what kinds of other contents to be annotated.

– ***Identify services and other contents to be annotated.***

Please 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. Therefore, we focus on helping users to annotate the existing WSDL definitions to get SA-WSDL definitions. SA-WSDL [6] is one of the latest W3C recommendation for Semantic Web Service.

– ***Annotate – automatically and manually***

CA can annotate the legacy contents either automatically or manually. Users can just click a button and CA then 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. Users can also manually select the text they want to annotate and link it to the proper concept from ontology. The most important component of CA is the Key Concept Identication Tool (KCIT) [4], which can automatically identify key concepts from software-realted legacy content intelligently (more than exact text match, like many other existing approaches). CA can also be configured to better adopt different use cases. For example, when preparing a document such as WSDL, we can configure CA so that the tags' processing is enabled.

– ***View and revise annotations***

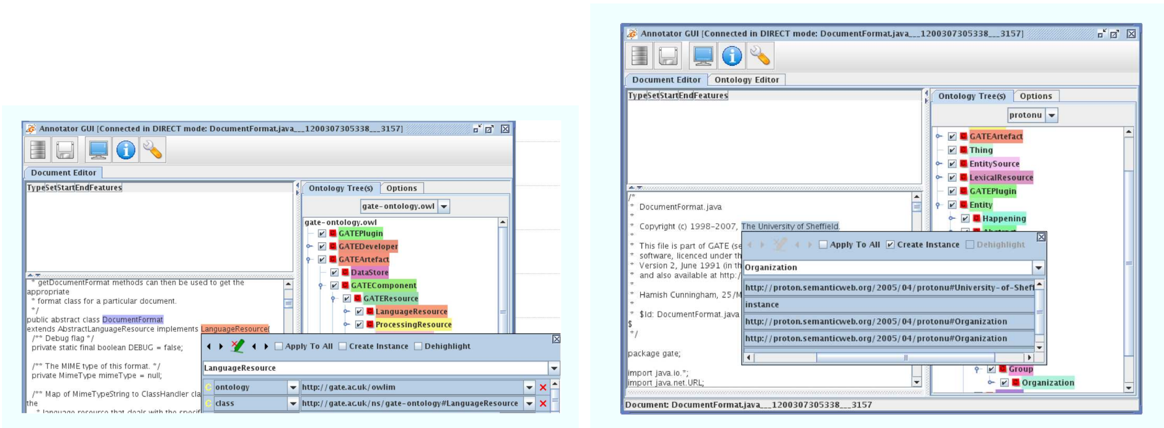
The results of the content augmentation process can be viewed graphically by the users as highlights over the original content . For example, Figure 4(a) shows the results of processing the GATEE class $DocumentFormat.java$ by the CA . The highlights are the semantic annotations created by the CA service, and the blue table at the bottom shows further details, in this case, of the annotation for $LanguageResource$ that refers to the class $LanguageResource$ from the GATE ontology. The resulting SA-WSDL descriptions for GATE services can be downloaded at `....`

The automatic annotating results could contain some flaws, and we need to ensure that these semantic metadata are correctly asserted. The annotation could be improper in several ways.

– Missed annotations. If the domain experts realize that there are some WSDL elements or texts in the legacy document, which should be annotated, but were missed

by CA, users can manually annotated them. If there is no proper concept within the existing ontology, new concept will be asserted into the ontology.
– Unnecessary annotations. It is possible that CA has put some unnecessary annotations. Domain experts have to delete those annotations manually.
– Annotations with wrong concepts. If domain experts realized that the concept CA has chosen for annotation is not the most suitable, they need to revise them.

Domain experts need to manually check the correctness of them. For example, as shown in Figure 4(b), the string *University of Shefeld* was not annotated as an *organisation*. To add this annotation to the document, rst the text is selected/highlighted, and then the relevant ontology resource is chosen from the hierarchy on the right or from the drop-down list of resource names in the dialogue. In this particular examples we are annotating the string University of Shefeld as referring to the class Organisation.



(a) Viewing results using Annotator GUI (b) Adding a new annotation

Fig. 4. CA interface

– ***Ontology population***

The CA can also identify a set of potential instances for the classes in the domain ontology from the legacy content. User will decide whether or not to accept these assertions.

During above processes, whenever the domain ontology is revised, users need to ensure the ontology is still correct.

Finally the legacy contents and the related semantic augmentations are stored in the heterogeneous knowledge store. This is important if users are working with large datasets. It is also the safest way to ensure that the annotations can be reloaded as same as before. The annotation can be saved either separately with the legacy contents or embedded within the legacy files.

In the SOA lifecycle, the next phases are service deployment and service descriptions evaluation and refinement

– ***Deploy Services***
– ***Evaluate and refine services***

Service Deployment refer to the process of deploying services within a service execution environment and service evaluation refers to the ongoing monitoring of a SOA system to determine whether it meets its design goals. During the course, if users realize any problems, the domain ontology and SA-WSDL definitions are revised. Because that those phases are not focuses of the TAO project (the scope of TAO is just generating the semantic descriptions), we will not give more details about them here. [2] presents some general guidelines for these tasks.

4 Methodology evaluation

The methodology presented in this paper has been validated in several high-profile case studies. For example, the GATE system, as presented in this paper, is a comprehensive open source platform (with thousands of users) and an aircraft maintenance application form Dassault Aviation is a data-intensive business process application (managing a multi-million business). For information about these case studies can be found at [1, 3].

The evaluation of TAO methodology and tools is conducted following the criteria proposed in [9]. We carry out the evaluation from several aspects:

What is the performance of the methodology and tools ontology extraction and annotation performance Time and effort to develop GATE ontology and annotation....

Is the extracted ontology a good basis for ontology building? expert evaluation The resulting ontology and annotation is satisfiable?

Does the extracted ontology and semantic annotated resources support a certain task, such as more effective Used for develop some applications. Query system? ...

5 Related works

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 [**?**,**?**]. While, Mariano et. al. [**?**] have identified seven of the most commonly used methodologies for designing ontologies from scratch, [**?**,**?**,**?**] 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 [9], 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 methodology for transitioning existing applications to SOA based on domain ontologies. 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 methodology 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. Furthermore, our methodology is fully supported by an integrated tool studio.

6 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 have taken an ontological view of Semantic Web Services. Both the lifecycle of SOA and building ontologies were examined, in order to understand the requirements for transitioning legacy systems to SOAs. This has been used for developing an initial methodology for the transitioning process based on domain ontologies learned from such applications. To support this methodology, a set of tools and a detailed cookbook style guide are presented as well. This transitioning process has be validated by a few large case studies.

Next, these tools will be integrated as a single suite which can provide a one-stop service to assist users to migrate their legacy applications to semantic enable services. More intensive evaluation is currently under performing.

Acknowledgements.

This work is partially supported by the EU-funded TAO project (IST-2004-026460).

References

1. 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.
2. Amardeilh F., Vatant B., Gibbins N., Payne T., Saleh A., and Wang H. H. Sws bootstrapping methodology. Technical Report D1.2, TAO project, 2008.
3. Cerbah F. Case study 2: Domain ontology building and semantic augmentation of legacy content. Technical Report D7.2, TAO project, 2008.
4. Bontcheva K., Damljanovic D., Aswani N., Agatonovic M., , and Sun J. Key concept identication and clustering of similar content. Technical Report D3.1, TAO project, 2007.
5. Bontcheva K., Roberts I., Agatonovic M., Nioche J., and Sun J. Gate case study: Requirement analysis and application of tao methodology in data intensive applications. Technical Report D6.1, TAO project, 2006.
6. Holger Lausen and Joel Farrell. Semantic annotations for WSDL and XML schema. W3C recommendation, W3C, August 2007. http://www.w3.org/TR/2007/REC-sawsdl-20070828/.
7. Grcar M. Ontology learning services library (v1). Technical Report D2.1-v1, TAO project, 2007.
8. Grcar M. Ontology learning services library (v2). Technical Report D2.1, TAO project, 2008.
9. Sabou M. Learning web service ontologies: an automatic extraction method and its evaluation. *Ontology Learning from Text: Methods, Evaluation and Application*, 2005.
10. S. McIlraith, T. Son, and H. Zeng. Semantic web services, 2001.
11. Marinova Z. Heterogeneous knowledge store. Technical Report D4.2, TAO project, 2008.

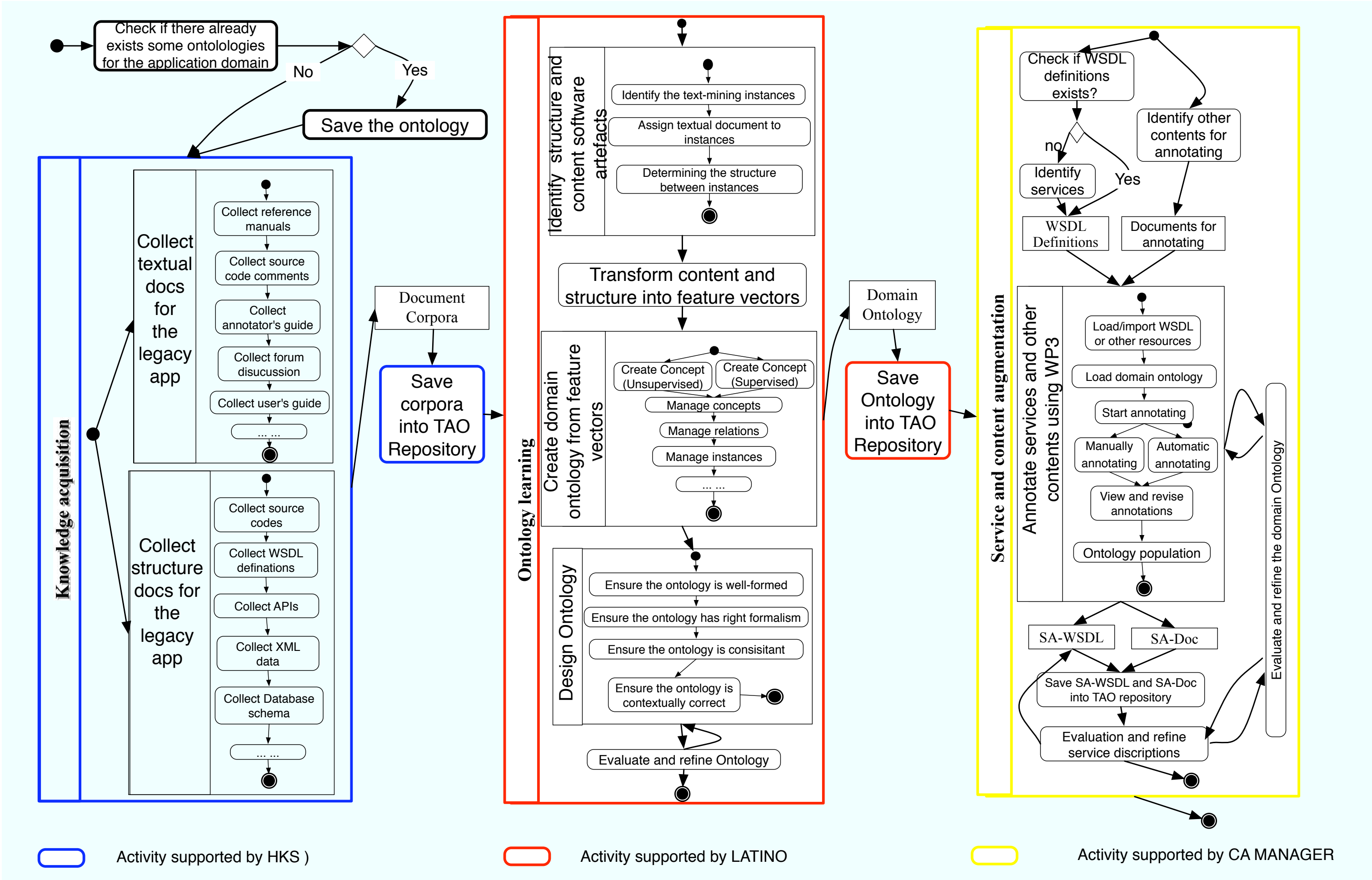


Figure 5. Cookbook methodology overview.