Transition of Legacy Systems to Semantic Enabled Application: TAO Approach

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 method and proper tool support, the transition 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 transition of legacy systems to Semantic Web Services platforms, and also tools to support such a strategy. In this paper we propose a method and its tool support, which allow users to migrate their applications to Semantic Web Services platform automatically or semi-automatically. The methodology is evaluated by the transition of GATE system as a case study.

keywords: Semantic Web, OWL, Ontologies, SA-WSDL, Service

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 [MSZ01] 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 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 transition 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 two main issues to be considered, namely: 1) Knowledge transformation dealing with the development of machine machine-accessible knowledge (ontology) about the legacy system suitable for service description, 2) Semantic Augmentation where the Web Service and other legacy documents are annotated using the relevant domain ontology. Without a systematic methodology and proper tool support, the transition 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 transition of legacy systems to Semantic Web Services platforms, and also tools to support such a strategy.

This paper proposes a new method and handy guidelines for addressing the above issues in particular Knowledge Accessibility and Augmentation, which in turn could lead to an automatic Platform Transformation. The main idea of the method is to identify the components/steps for creating web services to represent system 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 gap of interoperability and hence moving the system closer to SOAs. This method is part of Transitioning Applications to Ontologies (TAO) project [1] . TAO, which is in the European Sixth Framework Program, aims 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. We name the method presented in this paper as *TAO Method*.

The remainder of this paper is organized as follows. Section 2 provides a high level explanation of the method. Section 3 presents cookbook-style guidelines on how to adopt the methodology using the tools developed by TAO. Section 4 discusses the evaluation of the methodology and tools. Finally, Section 5 and 6 present the related work, conclusions of this paper and future work.

2 High-level transitioning process

The transitioning process outlined in this section is presented as a high-level composite lifecycle, which highlights the interactions between existing method-

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

ologies for developing Service-Oriented Architectures (which include Web Services and Enterprise Architectures), and for ontology design. We develop this abstracted lifecycle sketching of SOAs and building domain ontologies from text, and demonstrate how and where these should be linked.

Domain ontologies usually describe the conceptualization of entities, relations between them, 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* refers to the use of techniques for automatically or semi-automatically extracting ontologies from existing document corpora. *Ontology design* 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. The *ontology evaluation* process assesses whether or not the designed ontology is fit for purpose. *Ontology Refinement* refers to the refactorisation of the designed ontology to better represent the problem domain.

A typical design lifecycle for an SOA system may include phases like *Service Identification*, *Service Annotation*, *Service Deployment*, *Service Evaluation* and *Service Renement*.

For the aim of semantic enabled application transitioning, the TAO method 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 refine this to emphasis the contribution made by the description of an existing body of legacy application (for example, application 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.

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

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. 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.

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.

3 Methodology cookbook, tool support and case study

The abstract methodology presented in the previous section provides an high-level 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. Figure 1 shows the architecture of the transitioning environment. In it, the ontology learning tool is used to derive domain ontology from legacy application documentations (specifications, UML diagrams, code documentation, software manuals, incl. images). The content augment tool automatically identifies key concepts within legacy contents, going 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. In this section, we present a cook-book style guideline about the usage of TAO tools in transitioning processes. Note that this paper only focuses on providing a handy guideline about the usage of TAO tools, due to the limited space. For more technical details about the

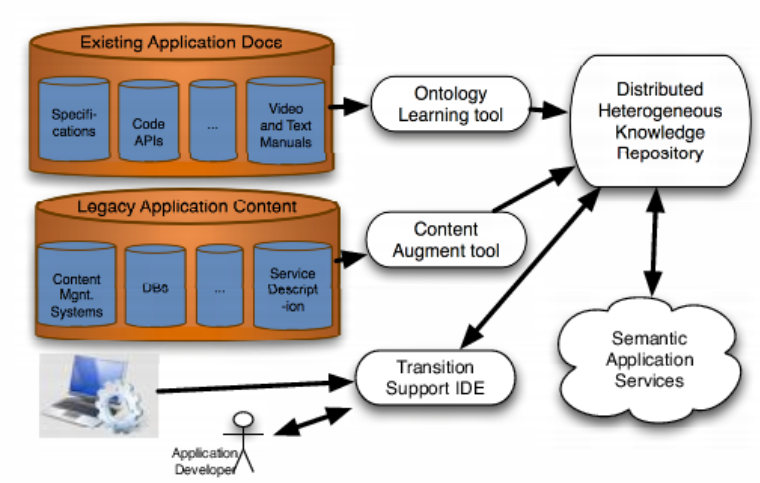


Fig. 1. Transitioning Process

various tool components, please refer to the respected cited technical reports.

To better illustrate the idea, we use the transition of GATE system as a case study. GATE [4] 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 it beomes difficult to understand, maintain and extend the system in a systematic way, due the large amount of heterogeneous information, which cannot be accessed via a unified interface. The advantage 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, users will be able to find 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. 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. As one of case studies used to evaluate TAO methodology, GATE exhibits all the specific problems that large software architectures encounter, which enables us to evaluate the methodology and tools intensively . [BRA+07] presents the detailed view of the advantages and possibilities arising from building domain ontology and application of semantic enrichment of software artifacts in the GATE case study.

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

3.1 Transitioning cookbook

As mentioned before, TAO method 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 4 presents a UML diagram to illustrate the main transitioning process and we explain major activities and the supported tools in details. All the TAO tools can be downloaded from `http://www.tao-project.eu/resources.html`.

Given a legacy application, 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 [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 NEON methodology [6] normally adopts and extend the found ontology directly. However, according to our past ontology developing experiences, this approach is not ideal for many use cases. The main reason is for most of time, it is difficult to find a perfectly matched ontology for a legacy application. Directly re-using complex domain ontologies built for the domain in similar projects could be a tedious task. The more complex an ontology is, the more tied it is to its original context of development and use and the less likely it is to fit another context. Its 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 ontologies 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 be built. In our methodology, if a related ontology is found (either directly existing 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 existing software artifacts. For the GATE case, we develop the ontology from scratch with the assitance of TAO tools.

3.1.1 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.

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

– Collect data resources

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 be related to the description of a legacy systems and their classification, please refer to the technical report [AVG+07]. For the GATE case study, first Java source code and 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 technical information about this heterogeneous knowledge store can be found in [Z.08].

3.1.2 Ontology Learning

The purpose of ontology learning from software artifacts 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. We have presented a detailed review of different ontology learning approaches in the technical report [AVG+07] and [GPMM04] also reviewed the major methods for semi-automatically building ontologies from texts. Due to the limited space, we omit these survey.

In this paper, we show how to learn domain ontologies based on the TAO scenario. *LATINO* [7] , as a part of the TAO Suite, is used for Ontology Learning purpose. LATINO is a more-or-less a 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 applications and their inter/intro-relationships are carefullly studied and made use of in the ontology construction process. We in-

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

troduce the term "application mining" which denotes the process of extracting this knowledge. Secondly, LATINO is not only limited to texts data sources. The usage of LATINO is summarized in this section.

In the previous step,a set of related data resources that describe the legacy application is collected. To get ontologies from these resources using LATINO, 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 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 as it depends on the available sources. Some potential choices include Java/C++ classes, methods, Database entities and the like. 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, 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 need to 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 technical information about those types of links and the different calculations of link weight can be

found in the report [GMG+07].

Create 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. 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 [Grc08], 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, we refer reader to the report [Grc08].

These feature vectors are further used as an input for OntoGen [8] 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 labeling the instances.

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

On each step the system asks if a particular instance belongs to the concept. 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 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, 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 2. This figure 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.

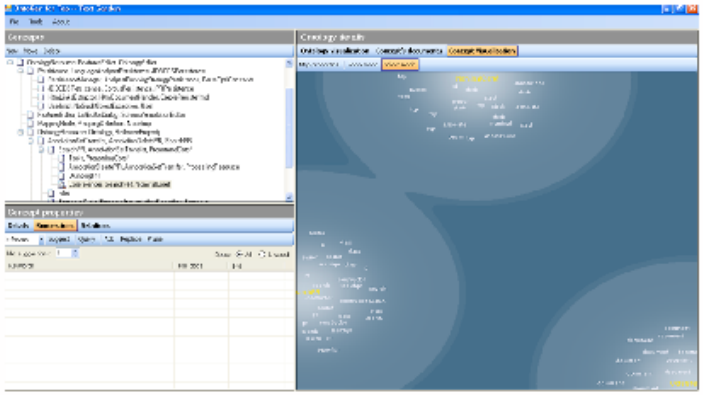


Fig. 2. Concepts derived from GATE source code using OntoGen.

Apart from concept identification, OntoGen/LATINO also implicitly infers subsumption relations (i.e. subClassOf) 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 [GMG+07].

Design Ontology

An important point to make is that the automated methods are not intended to extract the perfect ontology and 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 general, the resulting ontology should be designed to 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 the sake of feasible 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, this does not necessarily imply that it represents the real world accurately. 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.

The next step is to augment the existing content of a legacy application (including the service definition) semantically. We present the details in the following subsections.

3.1.3 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. TAO has developed a tool named *Content Augmentation Tool (CAT)* to assist users to annotate heterogeneous software artifacts automatically (semi-automatically). In essence, *CAT* 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 technical information about *CAT* can be found at [BDA+07]. The main component of CAT is *Key Concept Identification Tool (KCIT)* which has been exposed as a Web service (CA service) and can be accessed at `http://gate.ac.uk/ca-service/services/CAService`. CAT, KCIT and CA service can be found on `http://www.tao-project.eu/researchanddevelopment/demosanddownloads/content-augmentation-software.html` and also in [BDA+07] and [DKV+08].

To use *CAT*, we first need to identity 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

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 already exists some methods and tools for the reengineering of user interfaces from a legacy application. For example, [MGG+95] proposed to use data-flow analysis, [BCF03,SERIS03] proposed to use State Transition Diagrams and [Moo98] adopted techniques derived from artificial

intelligence to derive service interfaces. [MM00] described an architectural restructuring aimed at migrating character-oriented user interfaces to a web based front end. [Kit,Axi] provide tool kits to automatically generate WSDL files from implementation codes.

Here we mainly focus on helping users to annotate the existing WSDL definitions to get SA-WSDL definitions. SA-WSDL [LF07] is one of the latest W3C recommendation for Semantic Web Service.

Annotate – automatically and manually

As mentioned before, the main tool for performing Content Augmentation automatically is KCIT. 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. 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 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.

View and revise annotations

The results of the content augmentation process can be viewed graphically by the users as highlights over the original content inside the GATE GUI. For example, Figure 3(a) shows the results of processing the GATE class $DocumentFormat.java$ by the CA service . The highlights are the created semantic annotations, 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.

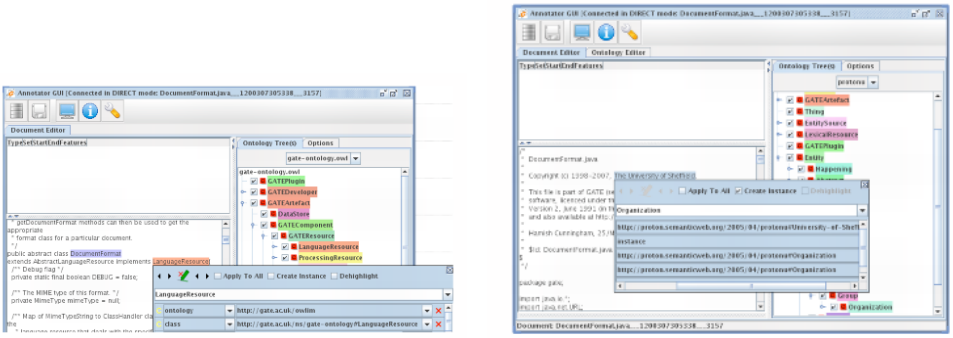
The automatic annotating results could contain some flaws, and we need to ensure that these semantic metadata are correctly asserted. The annotation could be improper in several ways.

- 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 service, users can manually annotate 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 service has created some

unnecessary annotations. Domain experts have to delete those annotations manually.

- Annotations with wrong concepts. If domain experts realize that the CA service has chosen the concept that is not the most suitable, they need to revise it.

Domain experts need to manually check the correctness of them. For example, as shown in Figure 3(b), the string *University of Sheffield* was not annotated as an *organisation*. To add this annotation to the document, first 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 Sheffield as referring to the class Organisation.



(a) Viewing results using Annotator GUI

(b) Adding a new annotation

Fig. 3. CA interface

Ontology population

Some of the Content Augmentation Tools 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 result of related semantic augmentation is 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 content or embedded within the legacy files.

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

Service Deployment refers to the process of deploying services within a service execution environment and service evaluation refers to the ongoing monitoring of an SOA system to determine whether it meets its design goals. During the course, if users encounter any problems, the domain ontology and SA-WSDL definitions are revised. As these phases are outside of the scope of the TAO project (the scope of TAO is just generating the semantic descriptions), we will not give more details about them here. [AVG+07] presents some general guidelines for these tasks.

4 Evaluation

In most cases, the migration methods and tools presented in the literature have been assessed in case studies. The drawback of this emphirical method is that determining trends and statistical validity is often difficult because each development is relatively unique. As a consequence, it is difficult to compare two different development profiles [ZW98]. Therefore, to get a more accurate evaluation, we validate TAO method 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 the *Dassault Aviation aircraft maintenance system* is a large database-intensive application. Other evaluation cases include *VIDEOLECTURES.NET system* [9] and *Amazon Web Service* [10] . We transition these systems into semantic enabled applications following the TAO method. More information about these case studies and their evaluation can be found in the technical reports [DKV+08,F.08,AVG+07]. In this section, some of the evaluation results for the GATE case study are presented.

Role within the team	**Number of Subjects**	**GATE Experience**	**Semantic Web Experience**	**TAO Experience**
Manager	... years	...	...	none
Research Fellow	...	...	...	none
PhD Students	...	...	...	none
other ...	...	...	...	none

Table 1
Experiment Subjects

In order to systematically evaluate how efficiently TAO process can assist users the transitioning process, the international standard *ISO 9126*, one of the state of the art software quality models, is adopted to measure different aspects of the method. *ISO 9126* has been widely used by developers, and

[9] `http://videolectures.net`
[10] `http://www.amazon.com/AWS-home-page-Money/b?ie=UTF8&node=3435361`

software quality reviewers to check the completeness of a definition and identification of quality requirements, design goals, test and criterions to ensure the quality of the software product. *ISO 9126* defines a quality model for software products by categorizing software product attributes into characteristics and subcharacteristics, such as usability, functionality, reliability, and the like. The measure of fulfillment of these characteristics is defined as the metrics for software qualities and it is normally done by means of customized questions in the context of the software entity to be evaluated. In order to assess the usefulness of TAO method and tools, their ease of use, and how much time engineers spent using them, a carefully designed survey questionnaire has been presented to those software engineers in the GATE team, whose background and the corresponding roles within the team are summarized in Table **??**. It is worth pointing out that none of the subjects were familiar with TAO migration strategy and the supported tools. The questionnaire is composed of questions expecting closed answers according to the Likert scale[Opp92]: from 1 (strongly agree) to 5 (strongly disagree). Table **??** presents some of the questions, which aims to access different aspects of TAO method according to *ISO 9126*. Subjects are asked to fulfill the questionnaire and also provide some explanations. More details about these questions and the response given by software engineers can be found in the technical report [MHB+08]. In this paper we only highlight a few interesting findings due to the limited space.

Learnability and Understandability: if users can learn TAO method easily and understand how to use it for particular task.		Average Score
Q1	I had no difficulties to learn TAO method.	
Q2	How much time (in terms of percentage) did you spend to learn TAO tools (A < 25% - B >= 25% - C >= 50%and =< 75% - D >= 75%)?	
Q1	I believe that in the ontology learning phase TAO methodology has provided useful guideline for determining text-mining instances?	
Q2	I find that the TAO cookbook readable and understandable in the transitioning process?	
Operability: if users can interact with TAO suit and control it easily		
Q1	In the knowledge acquisition phase, I believe it is easier (i.e. spending short time) to define and collect all the relevant resources about the legacy application following TAO method and using TAO repository?	
Q2	How many resources are needed for transitioning a legacy system using the TAO methodology?	

Table 2 – continued from previous page

Compliance/Conformance: If TAO method, tools and results are produced and specified according to some standards, style guides, usability conventions, UML notations, etc.		
Q1	I believe that the TAO methodology specified according to some standard notation?	
Q2	I believe that the resulted ontology is consistent and standard ontological language-compliance?	
Q1	I believe that the resulted ontologies are formalized in a proper ontology language description approach and described using the style guide (e.g. naming convention)?.	
Q1	I believe that the resulted SWS descriptions are formalized in a proper service annotation approach?	
Q2	I believe that Is the ontology obtained after the ontology learning process a good result in a task-oriented view and helps in the transitioning process.	
Q2	I believe that more common transitioning and modelling mistakes can be avoided using the TAO methodology.	
Q2	I believe that the SWS descriptions resulted from TAO a good basis for the service deploying.	
Q2	I believe that significant amount of time can be saved for transitioning a legacy system using the TAO method.	
Suitability: If TAO method and tool provide a set of appropriate functions for specific end-user tasks and objectives.		
Q1	I believe that the TAO method and tool cover all main phases in the semi-automatic construction of SWS descriptions resulted from the transitioning process of legacy applications.	
Q2	In the knowledge acquisition phase, I believe that the TAO method and tools have taken into account the necessary data sources types.	
Q1	I believe that the SWS descriptions resulted from TAO cover the objective and functionalities desired in the transitioning process.	
Attraction: If TAO gains the users, having into account for example the good results of some task, or thanks to save time and effort when user performs the tasks, etc.)		
Q1	I believe that the ontologies built using the TAO methodology has a better quality in contrast to the ontology built without using the TAO methodology.	
Q1	I believe that the extracted ontology a good basis for refining it afterwards.	
Q2	I believe that Is the ontology obtained after the ontology learning process a good result in a task-oriented view and helps in the transitioning process.	
Q2	I believe that more common transitioning and modelling mistakes can be avoided using the TAO methodology.	

Table 2 – concluded from previous page

Q2	I believe that the SWS descriptions resulted from TAO a good basis for the service deploying.	
Q2	I believe that significant amount of time can be saved for transitioning a legacy system using the TAO method.	
Q1	I believe that the ontology generated in the transitioning process covered the suffecient legacy system domain.	
Q2	I believe that the extracted ontology and semantic annotated resources support a certain task, such as more effective query and answer.	
Q2	In the service and content augmentation phase, I believe that the automated annotation provide an effective result (in term of missed or unnecessary annotations)?	

Table 2. Questions for TAO Method and Tool's Evaluation

4.1 Usage Evaluation

In general, software engineers report that they did not encounter significant difficulties with the methodology documentation, as after a very short training, they could use the TAO Suite without problems. Users also report to us that during the whole TAO transitioning process, the trickiest part is setting the weights for LATINO. This requires a lot of experimenting and testing, and there is no common rule to be followed in order to derive the most accurate hierarchy of concepts that can serve as a core for an ontology. The better the weight settings, the more precise concepts are derived and less time is left for the expert to check automatically derived ontology and manually perform changes. This matched our anticipation. To solve this problem, OntoSight software **??** is currently being developed which would speed up this process for future usage. This software will, given the golden standard ontology, provide means for automatically identifying which set of weights gives the best results, which will enable users to reduce time for experimenting with the weight settings, and changing them until the best results are achieved.

4.2 Ontology quality evaluation

The domain ontology plays the central role in the TAO method. A good quality ontology can significantly reduce users' workload in the following process. Most of ontology evaluation approaches proposed in the literature rely on domain experts' options and common sense. For example, **??** proposed to ask an expert ontology engineer to model a "gold standard" for the task and compare it with the generated ontology. **??** suggested to let domain experts to use the ontology

in an application and then evaluate the results. **??** define a set of ontology criteria, which need to be accessed manually by domain experts, based on common sense and domain knowledge.

To measure the quality of the GATE ontology that is created by the TAO method, apart from questioning engineers about their general opinion about it (by questionnaire), we wanted to know how much of the questions posted on the GATE mailing list can be actually answered using the developed ontology. 36 questions are randomly collected from the GATE user mailing list, where the GATE users enquire about various GATE modules, plugins, processing resources, and problems they encounter while using these modules. After examining these questions, they identified that out of these 36 questions, 61.1% (22 questions) were *answerable*: the GATE domain ontology that is developed following the methodology described in this paper, contained the answers for these questions. For the rest of the 14 questions (38.9%), the answer was not in the knowledge base. This is due to the questions that were usually enquiring about too specific details such as: *Can someone send me the codes for all characters that are classified as DEFAULT_TOKEN?* or *What is the correct location of the lexicon files for Hepple Tagger?*. Another set of questions for which answer was not in the knowledge base enquired about specific features that were usually not included inside user manuals and documentation, but were related to the knowledge of experienced GATE users, e.g. the familiarity with GATE GUI, or types of parsers for language processing available in GATE. For example, *Which Gate plugin parsers can find object and subject of a sentence?* or *Is it possible to see the POS tagging from the GATE GUI?*. Another set of queries were enquiring about Processing Resources, that have not been yet included in the GATE knowledge base due to various reasons (i.e. they have been included in the GATE distribution recently). This is when the Ontology Refinement phase of our methodology comes to place: the ontology need to be refined if needed and enrich the knowledge base so that 100% of *answerable* questions could be reached.

4.3 Performance Evaluation

The performance evaluation of the TAO method can be expressed through the evaluation of the TAO Suite, more specificaly through the evaluation of its components. The two most important TAO Suite components are *Ontology Learning tools* and *Content Augmentation Tools*.

However, it is very difficult to report on any quantitative evaluation regarding the Ontology Learning tools, as it was difficult to measure exactly the time spent for ontology acquisition, due to the nature of this process. However users do report back to us that

> ... TAO ontology extraction phase is generally automatic and lowers the cost of bootstrapping the domain ontology significantly, and also makes it easier not to miss important domain concepts. ....

The evaluation of Content Augmentation tools is reported through the performance of the KCIT tool. As KCIT is primarly the semantic annotation tool, we have measured its performance using standard information retrieval measures, namely *precision* and *recall*.

In order to prepare the experiment, we first collected the GATE software artefacts of various types, namely source code, source documentation, GATE user manual, publications and forum posts from the GATE mailing list. Next, we needed to configure the KCIT in order to get desired performance.

4.3.1 KCIT configuration

To configure KCIT, we have randlomly selected 3 documents and run the KCIT using default settings, to produce annotations based on the GATE domain ontology. Default settings of KCIT are based on the extraction of all ontology resources, their properties and the property values. However, in some cases, such as the case of the GATE case study, some datatype properties usually have numbers as values, in which case produced annotations can be wrong. For example, each *Processing Resource* in GATE, has some set parameters, and the default value for these parameters is often a number, e.g. 1. However, when annotating the GATE software artefacts, it is rarely the case that the number 1 is referring to the default value of the resource parameter, but it is often used, e.g. for section numbers, etc. Therefore, we need to configure KCIT to exclude the datatype properties whose values could lead to ambiguous, and often wrong annotations. After the examination of the annotated documents, we have decided to exclude the following properties when running KCIT:

- `gate:parameterHasDefaultValue`: as mentioned in the previous example, we have noticed that the value of this property is often a number, which causes producing wrong annotations [11] ;
- `gate:parameterHasName` and `gate:resourceHasName`: values of these two properties are often the same for a huge number of ontology resources, therefore creating enourmous number of overlapped annotations, which make it difficult to filter and resolve ambiguity. For example, a *Processing Resource* in GATE usually has the parameter with name *document*, which resulted in having more than 100 annotations created whenever string *document* occured in the text.

[11] 'gate' is used instead of the full namespace which is 'http://gate.ac.uk/ns/gate-ontology#'

- `gate:parameterHasComment`: the value of this property is usually a long sentence explaining details about the specific ontology resource. However, these details are rarely useful during the process of annotation, as it is rarely the case that the exactly same sentence would appear in the text.

4.3.2 Running KCIT: results

We have selected 20 documents to serve as a representative corpus of the GATE software artefacts. Among these 20, we have choosen various types of documents, among which there were:

- 4 forum posts from the GATE mailing list
- 3 java classes from the GATE source code
- 7 chapters of the GATE user manual
- 3 publications about GATE
- 2 Web pages accessible from the gate.ac.uk site
- 1 GATE application developers guide

This selection is made in order to cover not only the knowledge about GATE, but also different types of documents about GATE software artefacts. We have first manually annotated these documents to create a golden standard corpus. Next, we have run the KCIT over the clean corpus to automatically annotate it, and compared results using GATE Benchmarking tool. Results are shown in Table 3.

	Precision	**Recall**
4 Forum posts	98.6111111	100.0
7 chapters of the GATE User Manual	96.0007672	96.9611548
2 Web pages	94.379845	95.0787402
3 publications	89.796798	95.8392268
3 java classes	97.2592593	98.8636364
1 GATE application developers guide	96.484375	98.40637450199203
Total	94.28304627516082	96.87633262260128

Table 3
Precision and recall measures for the selection of the GATE software artefacts

For the 20 documents selected to represent the GATE software artefacts, 4523 created annotations were correct, 41 annotations were partially correct, 126 annotations were missing, and 255 were spurious.

Further inspection of the annotated documents revealed that the majority of wrong/missed annotations were due to the malfunction of the *Morphological*

Analyser used. For example, the analyser could not extract the root of the words correctly in cases when the plural of acronyms (or the plural of camel-Cased words) was used, for example, extracted root for *LanguageResources* remained *LanguageResources*, as well for the *PRs* – the root remained *PRs*.

In addition, plenty of annotations for word *learn* were created, although just a minority were correct. The reason is: the KCIT does not rely on context, and is not smart enough to decide wheather something is relevant or not based on the context. On the contrary, each appearance of the word *learn*, even in the sentence 'learn GATE using movie tutorials' was annotated as referring to the plugin called *learning*. Similarly, many annotations were created for each occurence of *resources*, even if this was not referring to the GATE resources. For example, when reporting the problems on the mailing list, a user said 'I cannot waste too much resources on the server...', where *resources* definitely are not the GATE ones. At the moment, KCIT is not capable of distinguishing between such words. However, detection of rarely used and very domain specific terms is very reliable.

In addition, some overlapped annotations were not filtered properly by KCIT, and affected the overall performance. For example, the occurence of *Stemmer PR* was annotated so that *Stemmer PR* refers to the *Processing Resource* with name *Stemmer PR*, which is correct. The part of this annotation, namely *PR* was annotated as well, to refer to the class *Processing Resource*. However, although correct, this second annotation should be removed during the filtering phase of the KCIT, as it is redundant. Similarly, *ANNIE NE Transducer* was annotated so that the whole string refers to the processing resource with this name, while the *Transducer* is annotated to refer to the *JAPE Transducer*, which should be removed during the KCIT filtering phase.

Overall, we can conclude that the performance of the KCIT is satisfiable especially for the cases of smaller documents. For example, in cases of forum posts or java classes, annotations were produced with a very high precision, and specifically recall. As documents get bigger, such is the case for publications, the performance is degraded, but still remaining at a reasonable level.

5 Related work

Several methods and tool have been presented in the literature for transitioning legacy systems [BCF03,BS95,CCdLL98], and also for the migration to web technologies [LFM+07,CFFT06,SS03]. Those methods focused more on various general issues which need to be considered for any software engineering project (e.g. target system development, testing, and database model selection). Other issues, which are specific to migration, include target system

cut-over with mission-critical support, target database population and etc. Depending on different situations, users can choose from various approaches:

Wrapping involves developing a software component called wrapper that allows an existing software component to be accessed by other components which need not be aware of its implementation.

Re-development also referred as the Big Bang or Gold Turkey approach involves re-developing a legacy system from scratch using a modern architecture, tools and databases, running on a new hardware platform.

Migration allows legacy systems to be moved to new environments that allow information systems to be easily maintained and adapted to new business requirements, while retaining functionality and data of the original legacy systems without having to completely redevelop them.

By contrast, our paper focuses on a more specific task, i.e.,aiding users in the process of transitioning of legacy applications to semantics-based SOA, via ontologies and refactoring. Among which, two most vitals are developing a domain ontology based an application system and method to describe the functionalities of the service effectively using the ontology.

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 [JBCV,GPMM04]. While, [FLGPE$^+$02] has identified seven of the most commonly used methodologies for designing ontologies from scratch. [Gru93,SPKR96] 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 methodology for transitioning existing applications to SOA based on domain ontologies. For example, [LFM$^+$07] 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 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.

6 Conclusion and future work

A key requirement for 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 cook-book style guide are presented as well. This transitioning process has been 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 enabled services. More intensive evaluation is currently under performing.

Acknowledgements.

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

References

[AVG+07] Florence Amardeilh, Bernard Vatant, Nicholas Gibbins, Terry R. Payne, Ahmed Saleh, and Hai H.Wang. Sws bootstrapping methodology. Technical Report D1.2, TAO Project Deliverable, 2007. http://www.gate.ac.uk/projects/tao/webpage/deliverables/d1-2.pdf.

[Axi] Apache Axis. http://ws.apache.org/axis.

[BCF03] Diego Bovenzi, Gerardo Canfora, and Anna Rita Fasolino. Enabling legacy system accessibility by web heterogeneous clients. In *CSMR '03: Proceedings of the Seventh European Conference on Software Maintenance and Reengineering*, page 73, Washington, DC, USA, 2003. IEEE Computer Society.

[BDA+07] Kalina Bontcheva, Danica Damljanovic, Niraj Aswani, Milan Agatonovic, James Sun, and Florence Amardeilh. Key concept identification and clustering of similar content. Technical Report D3.1, TAO Project Deliverable, 2007. http://www.gate.ac.uk/projects/tao/webpage/deliverables/d3-1.pdf.

[BRA+07] 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.

[BS95] Michael L. Brodie and Michael Stonebraker. *Migrating legacy systems: gateways, interfaces & the incremental approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[CCdLL98] G. Canfora, A. Cimitile, A. de Lucia, and G. A. Di Lucca. Decomposing legacy programs: A first step towards migrating to client-server platforms. In *IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension*, page 136, Washington, DC, USA, 1998. IEEE Computer Society.

[CFFT06] Gerardo Canfora, Anna Rita Fasolino, Gianni Frattolillo, and Porfirio Tramontana. Migrating interactive legacy systems to web services. In *CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering*, pages 24–36, Washington, DC, USA, 2006. IEEE Computer Society.

[DKV+08] 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.

[F.08] 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.

[FLGPE+02] 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`.

[GMG+07] 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.

[GPMM04] 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.

[Grc08] 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.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[JBCV] D. Jones, T. Bench-Capon, and P. Visser. Methodologies for ontology development.

[Kit] Emerging Technology Tool Kit. http://www.alphaworks.ibm.com/tech/ettk.

[LF07] 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/.

[LFM+07] 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.

[MGG+95] Ettore Merlo, Pierre-Yves Gagné, Jean-Francois Girard, Kostas Kontogiannis, Laurie Hendren, Prakash Panangaden, and Renato De Mori. Reengineering user interfaces. *IEEE Softw.*, 12(1):64–73, 1995.

[MHB+08] Jesus Martin, German Herrero, Kalina Bontcheva, Danica Damljanovic, and Farid Cerbah. Case study reports on evaluating the methodology. Technical Report D1.3, TAO Project Deliverable, 2008. http://www.gate.ac.uk/projects/tao/webpage/deliverables/d1-3.pdf.

[MM00] Melody M. Moore and Lilia Moshkina. Migrating legacy user interfaces to the internet: Shifting dialogue initiative. In *WCRE '00: Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, page 52, Washington, DC, USA, 2000. IEEE Computer Society.

[Moo98] Melody Marie Moore. *User interface reengineering.* PhD thesis, Atlanta, GA, USA, 1998. Director-James D. Foley and Director-Spencer Rugaber.

[MSZ01] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[Opp92] A. N. Oppenheim. *Questionnaire Design, Interviewing and Attitude Measurement.* Pinter Publishers, 1992.

[SERIS03] E. Stroulia, M. El-Ramly, P. Iglinski, and P. Sorenson. User interface reverse engineering in support of interface migration to the web. *Automated Software Engg.*, 10(3):271–301, 2003.

[SPKR96] 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.

[SS03] Harry M. Sneed and Stephan H. Sneed. Creating web services from legacy host programs. *5th International Workshop on Web Site Evolution*, 0:59, 2003.

[Z.08] 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.

[ZW98] Marvin V. Zelkowitz and Dolores R. Wallace. Experimental models for validating technology. *Computer*, 31(5):23–31, 1998.

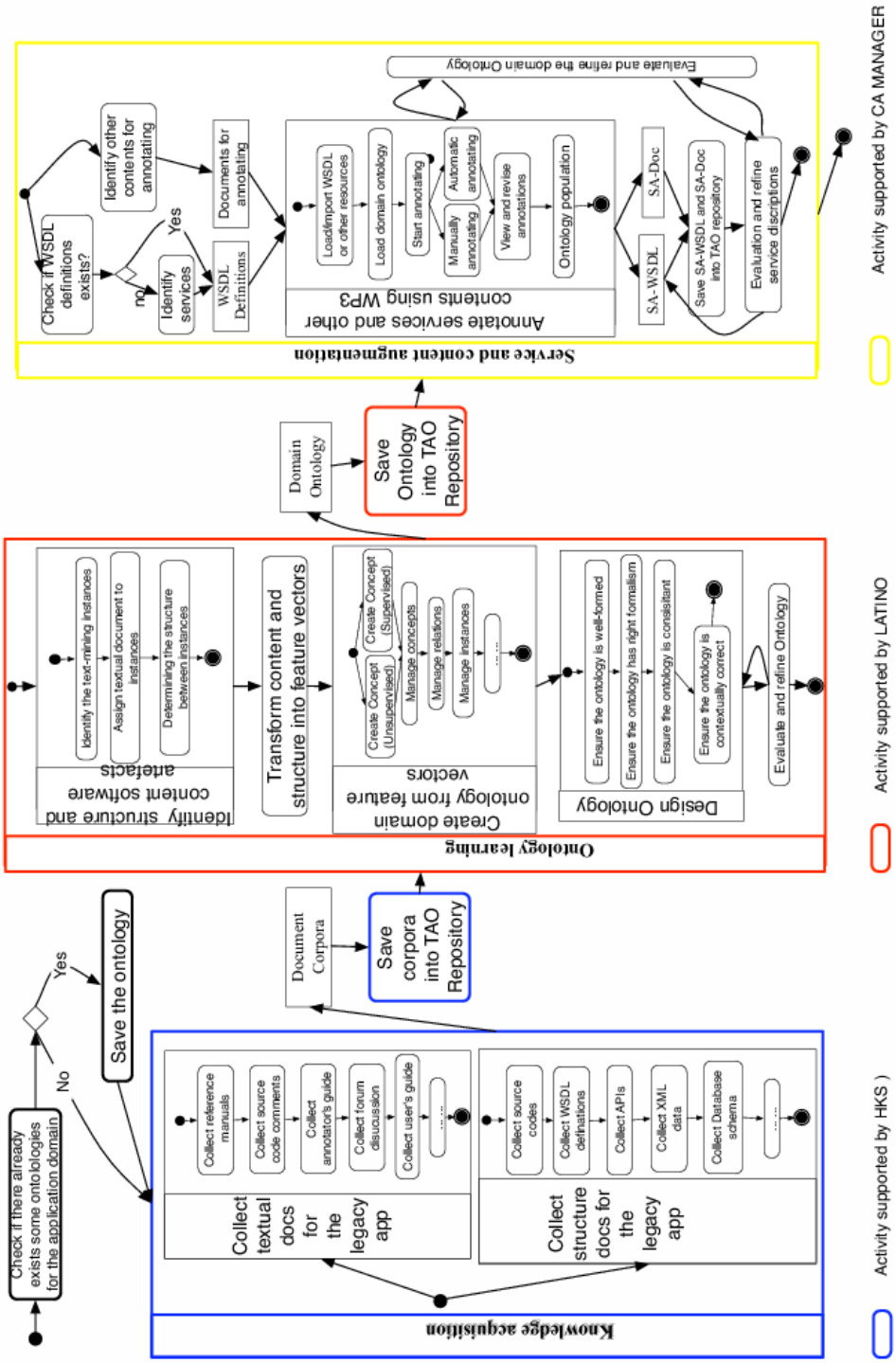


Fig. 4. Cookbook methodology overview.