Ontology-enabled Documentation of Service-oriented Architectures with Ontobrowse Semantic Wiki

Hans-Jörg Happel[1], Stefan Seedorf[2], and Martin Schader[2]

[1] FZI Research Center for Information Technologies, Karlsruhe
happel@fzi.de
[2] University of Mannheim, Chair in Information Systems III
{seedorf|schader}@wifo3.uni-mannheim.de

Abstract: Documenting and maintaining an enterprise-wide service-oriented architecture (SOA) causes a substantial management effort, which should be addressed by intelligent, scalable solutions. A particular challenge is that business experts, developers, and software architects take different perspectives on a SOA, each favoring various description formats and sources, which leads towards a scattering of architectural information. Ontobrowse Semantic Wiki specifically addresses these issues by providing an ontology-based integration and documentation platform for architectural information. In this paper, we identify key issues arising from documentation and maintenance of a SOA by introducing the case of an insurance company. We give a detailed description of the Ontobrowse approach and its implementation, and explain how ontologies, artifact mappings, and architectural rules are created to support the Enterprise SOA case.

1 Introduction

The paradigm of service-oriented computing has lifted the development of business applications to a higher level of abstraction. Instead of thinking in technical categories like components or objects, software functionality is bundled in services, which correspond to business operations of the organization. Complex workflows can be realized by aggregating functionality from simple services. A concrete software infrastructure implementing this paradigm is called a service-oriented architecture (SOA) [HS05].

In an organization adopting a SOA, the standard working processes change for the multiple stakeholders involved, i.e., service developers, business experts, and software architects. First, service developers have to think in specification terms rather than taking an implementation view. Since services are black-box specifications, which hide the details of the internal realization, metadata describing their properties is crucial. In an enterprise-wide scenario, it therefore has to be documented which services are available, where the services are being deployed, how they can be invoked, and who is responsible for them.

Second, business experts are interested in available business functionality and operational efficiency. Since service-orientation leads to a rising level of alignment

between business processes and IT implementation, it is important to monitor and guide the development of the service landscape according to changing business requirements.

Third, software architects are interested in ensuring that services are specified and composed in such a way that key quality attributes such as performance, reusability, and modifiability are met. In order to achieve this, architectural patterns, rules, and policies are defined at the beginning of a SOA project. These are subsequently rolled out in service design decisions. However, as a SOA grows more complex, architects find it difficult to continously monitor if the current service definitions comply with the set of architectural rules.

Since all these perspectives on a SOA are tightly interwoven, it is desirable to provide an integrated view and access to different aspects of SOA documentation and maintenance, which allows the different stakeholders to incorporate change requests more rapidly and thoroughly and to better oversee consequences of their own actions. However, achieving this is challenging from both a technical and a functional point of view.

Technically, while a considerable number of standards to describe service properties like interface, behavior, and orchestration is available, this information is scattered in disjoint information spaces, which makes it hard for stakeholders to get an overview. Functionally, this is due to the fact that most SOA artifacts emerge “bottom-up” driven by the requirements of the respective tool suite of each stakeholder.

Thus, to align the different perspectives, a “top-down” layer is required, which serves as an integration and reference point for previously scattered information. In order to address the issues, we have developed Ontobrowse, a semantic wiki based on ontologies, which provides an infrastructure to extract knowledge from external sources [HS07]. In the case of an enterprise SOA, it can serve as single point of information, which allows to integrate different information from different stakeholder perspectives, thus covering both technical and business aspects of a SOA.

The remainder of the paper is structured as follows: In section 2, we present an Enterprise SOA case and derive the requirements for our documentation tool. We also address the shortcomings of existing approaches. In section 3, we shortly introduce the basic elements of our solution, namely ontologies and semantic wikis. In section 4, the architecture and realization of Ontobrowse is described. Section 5 demonstrates the setup with a concrete SOA ontology, artifact mapping an architectural rule to support the requirements. Finally, we summarize our findings in the conclusion.

2 Documenting and Maintaining an Enterprise SOA

In this section, we introduce the case of an enterprise SOA in an insurance company. It characterizes the systems, actors, and development artifacts in a concrete SOA environment. This helps us to identify the shortcomings of current SOA documentation practices and derive requirements for a suitable tool support.

2.1 Enterprise SOA Case

InsCorp Inc. is an insurance company, which has established itself as a top ten player for life insurance in its domestic market. The system landscape at InsCorp has continuously grown over the last decade and become highly heterogeneous. It consists of several legacy systems as well as databases, enterprise applications, ERP, and CRM systems. One major challenge is aligning the IT landscape to changing business requirements. In particular, cooperation with third-party vendors and product diversification has led to a high number of client systems with similar functionality. The current enterprise architecture makes managing change more and more difficult. In order to allow for further growth, InsCorp has commissioned ITCorp Inc. with a new project. The purpose is to develop a new enterprise-wide SOA, which meets the needs of its agile business environment.

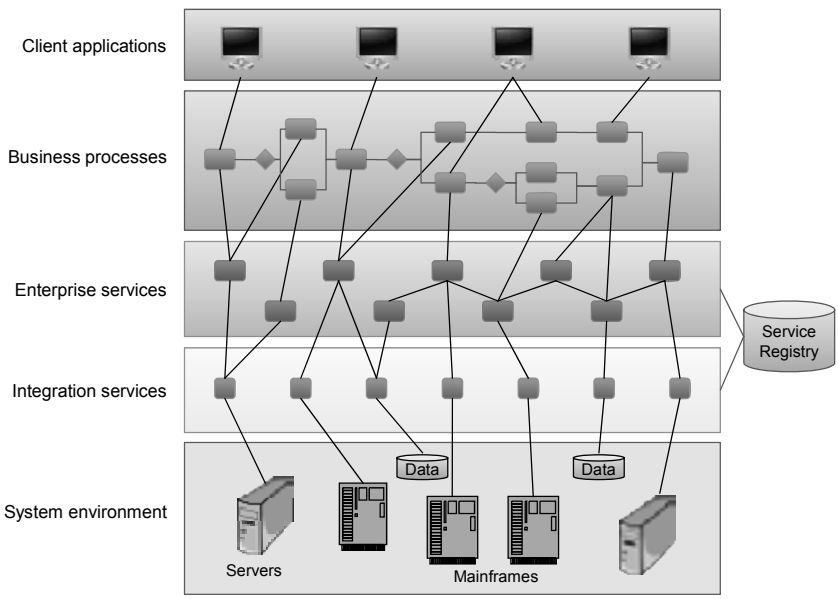

Figure 1: Organization of services in architectural layers

The development and evolution of a new architecture involves a number of different stakeholders who collaborate throughout the SOA lifecycle. The service architecture is organized in several logical layers as depicted in Figure 1. Different tasks and roles can be assigned to each layer. The *system environment* layer is maintained by several groups in the IT department. In the *integration services* layer, the functionalities of individual systems are exposed as Web Services to achieve technology abstraction. Maintainers of legacy systems and SOA architects have to cooperate in order keep these two layers in sync. The *enterprise services* layer aggregates basic services from the integration layer to support business activities. Business experts, process engineers, and software architects have to work seamlessly together to specify enterprise services and business objects. In the *business process* layer, process engineers realize workflows, which are

composed from the enterprise layer. Finally, application frontends in the *client application* layer execute these business processes to perform a business task. This top layer drives the evolution of the SOA, since the frontend applications require different business processes and data formats. Changing end user requirements can thus require modifications in all further layers.

Such modifications in turn result in considerable coordination overhead, since most of the layers depicted in Figure 1 are maintained by different stakeholders who stem from different organizations or at least different departments inside InsCorp.

These stakeholders document their views using different notations and formats. At InsCorp business analysts, responsible for *client applications*, use very generic tools such as Word and Excel for the functional description of services. In contrast, process engineers and service developers, responsible for the *business processes* and *enterprise services* layer, primarily work with technical specifications. Services are described in WSDL, the business objects are defined in XML Schema. Process engineers use a visual editor, which generates executable business processes in BPEL. Runtime information is captured using a custom-made service registry. At InsCorp, these tasks are completely managed by employees from ITCorp. Inc. The *integration services* and system level is partly maintained by technical departments of InsCorp., while a number of legacy systems is managed by different external contractors with specialized expertise.

Because various stakeholders contribute to the enterprise architecture, new challenges concerning architecture documentation and maintenance arise. Various issues stemming from this complex dependencies are reported by InsCorp. First, propagating changes across different layers involves considerable coordination effort and time. Different tools and data formats are used by various stakeholders, which requires many hand-crafted transformations. This is a frequent source of problems, especially if staff members vary, since people working at adjacent layers do not maintain any shared conceptualization of their collaboration beyond the technical artifacts and descriptions they exchange.

Furthermore, SOA artifacts are maintained in separate information spaces, which makes it difficult for other stakeholders to find relevant information for a task at hand. Also, much information gets lost at the interfaces between different layers, which leads to communication gaps– e.g., between business experts and developers– when there is no representation for the mapping between functional and technical service descriptions. While for example the response times of certain legacy systems might be important for a business analyst to cast a decision about some functionality, there is no place in the current setting, where such information could be stored.

From the perspective of InsCorp, this is highly problematic, since it creates a high dependency on the staff and work performed by ITCorp. Furthermore, it is neither possible to easily check the consistency of the overall SOA architecture, nor to estimate modification efforts in dependent layers, if changes are necessary. This inhibits InsCorp's ability to innovate and quickly react on market demands.

Although the original purpose of the SOA project was to reduce complexity and foster reuse, the large number of services and business processes makes it difficult to get an

overview of the architecture. Therefore, InsCorp and ITCorp look for a documentation tool, which integrates the various architectural views and which can be customized to project-specific needs.

2.2 Requirements

At first glance, the introduction of an enterprise-wide SOA simplifies the task of architecture documentation and maintenance due to a higher degree of standardization. However, as multiple stakeholders are involved a SOA also leads to a scattering of architectural information. Thus, a documentation tool should fit for different stakeholder viewpoints by creating an integration space.

One possibility for that is to go for a heavyweight, integrated SOA management suite, as it is offered by several vendors. However, these suites typically require complete migration of existing solutions and enforcement of a strict process across the whole architecture. For InsCorp, this is hardly possible, due to the complex organizational and technical ecosystem, which is already in place. As these stakeholders use dedicated formats and tools to describe their viewpoint a SOA documentation tool should be able to integrate various types of architectural information from existing environments, e.g., service specifications and their functional descriptions.

Considering this situation, we propose a lightweight, non-invasive solution that integrates as seamlessly as possible with the methods and tools already in use. Such a tool should neither replace nor interfere with existing tools and workflows, but provide a complementary, cohesive point of reference for all stakeholders participating in the development and maintenance of the SOA.

From the case in the previous section three main functional requirements for such a tool can be identified. These will be discussed in more detail below (c.f. Figure 2):

- Searching and browsing of SOA elements
- Checking the consistency of SOA elements
- Text documentation of SOA elements

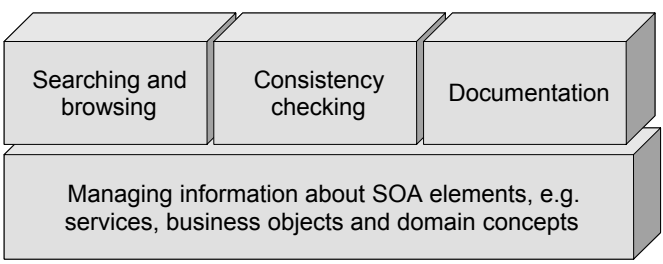

Figure 2: Overview of requirements

Searching and browsing of SOA elements is required to allow a better overview of the SOA for all stakeholders, e.g., architects, developers, and business experts. By describing the precise relations between services, business objects, and domain concepts, it is possible to gain a quick overview, e.g., which services require or return a certain business object. This enables an easy access to explore the list of available services, which may also promote the reuse of existing services.

Another requirement is consistency checking of SOA elements. What is meant here is not the syntactic consistency of the SOA, which is checked by the development-level tools, but consistency on a semantic level. Those are typically expressed as architectural and design rules, e.g., “Services may not call other services more than one layer apart” or “Services should not have more than five operations”.

Finally, the approach should enhance service documentation. Normally, there are two kinds of service documentation: technical descriptions, which are maintained by developers and business-oriented documentation residing in separate documents. This situation makes it difficult to get the complete information about a service, since documentation is distributed across physical storage locations and media types. The purpose is to create a single point of information for every service and business object. These elements of a SOA should be accessible by specifying a URL. In addition to accessing information, it should also be possible to add text documentation directly to a SOA element.

Besides that, the system has to support the management of SOA elements, such as services, business objects, domain concepts etc. Since the Enterprise SOA case requires a flexible solution, which is adaptable to the environment of a particular project, it must be possible to import arbitrary architectural descriptions that are not maintained internally but externally using project-specific formats and tools. Further, we need administrative features for governing this process such as updating and deleting information that has been acquired from external sources.

2.3 Related Work

Documenting and maintaining complex architectures has been a problem in Software Engineering for a long time. But as we will see in the following, existing work does not fulfill all requirements, which we described in the previous section. While there are systems and approaches, which support either searching and browsing, consistency checking or documentation for particular application scenarios, it will turn out that they lack flexibility to deal with both structured and unstructured information as it is required in a setting with diverse stakeholders.

Along with the increasing significance of SOA, new enterprise architecture tools have been emerging. These are usually geared towards managing the entire service life cycle, from analysis and design to versioning, deployment and monitoring. One shortcoming of these tools is that they lack openness since they are based on a relatively fixed metamodel. This may lead to problems when an organization deviates from a standard scheme, e.g., using other formats than WSDL or BPEL. In contrast, our solution is geared towards allowing maximum flexibility but at the same time enabling strong semantics. Since the stakeholders first have to agree upon one or more ontologies, the upfront investment setting up an environment can be higher. However, the costs can be reduced if existing ontologies and format mappings are reused.

As a SOA is only one specific instance of a software architecture, there are also more general approaches for enterprise architecture documentation. In most cases an enterprise architecture is represented using a number of different views (see [Kr95] for example). Depending on the particular view, formal or informal notations such as architecture description language (ADL) are used [MT00]. However, the case in section 2.1 substantially differs from the purpose of ADLs. Whereas ADLs focus on the specification and verification of a single view, we strive towards an integrated approach with a first-class representation for integrating all local views. Moreover, we want to include both formal and informal descriptions in this representation,

Universal modeling languages such as the Unified Modeling Language (UML) are also useful to describe a number of architectural views. Although the UML can be extended to cover various aspects, it does not include mechanisms for information integration and automated reasoning. The UML is also not useful for text documentation, which limits its applicability as the only representation language in the Enterprise SOA case.

Some researchers have proposed wikis for architecture and software documentation. Aguiar and David present a wiki-based approach to integrate heterogeneous software specification resources into a single document [AD05], while Bachmann and Merson investigate the advantages of wikis compared to other architecture documentation tools [BM05]. However, the approaches lack a formal model – information is managed in an unstructured way.

Moreover, formal ontologies have been proposed for architectural documentation [WF99] and for building "software information systems", describing the interrelationships of domain models and source code [We03]. These approaches are usually bound to a specific tool, which cannot be tailored to individual project needs or follow a very strict philosophy of software architecture.

To summarize, existing solutions for architectural documentation either impose a very strict formal model, which lacks the flexibility of documentation required in our case, or completely lack any formal model, which makes consistency checking and searching difficult. What is sought is an approach that allows to bridge varying degrees of structure.

Knowledge engineering, as a domain with similar problems, has recently bred the concept of "Semantic Wikis", which combine Wikis and ontologies [Vö06]. Since first applications in the domain of Software Engineering appear promising (see e.g. [De05]), we will investigate its suitability with respect to our requirements in the following section.

3 Foundations of the Ontobrowse Approach

In this section, we introduce the basic concepts of our solution. Revisiting our requirements in section 2.2, we regard ontologies and semantic wikis to be perfectly suited for our approach. Our goal is to model the structure of existing data (e.g., services defined in a WSDL file) and align these models to a top-level SOA ontology. At the runtime of our system, facts from existing artifacts can thus be automatically extracted and imported into our knowledge base. Modeling the information of the SOA domain and related development artifacts and processes in a SOA ontology will have the following benefits:

- Ontologies enable the integration of different aspects and data sources of the domain in question. By aligning the individual data source to a top-level ontology, the complete information can be searched and browsed in a unified way. Semantic links between different concepts (e.g., a process defined in a WSDL file and a user defined in an issue tracking system) can be modeled. Previously disjoint information such as e.g., the operations of a service and information about runtime behavior can thus be aggregated and presented in a single place (see e.g. Figure 7).

- Further, this integrated model of the SOA domain can be automatically checked for consistency. This can involve basic consistency checks, such as cardinality constraints (e.g., "a process should only have one owner") or more complex checks, which can be formulated using rule-based approaches (cf. section 5).

Finally, the ontologies and a knowledge base serve as backbone for a semantic wiki, where each entity in the knowledge base can be browsed, queried, documented, and semantically referenced.

3.1 Ontologies

An ontology in information systems (IS) provides the structure for a commonly agreed understanding of a problem domain. According to a widespread definition, "an ontology is an explicit specification of a conceptualization" [Gr93]. Ontology specifications have varying degrees of formalization; for example a shared vocabulary of terms can be regarded as lightweight ontology. Usually, ontologies are specified in a knowledge representation language using sets of concepts, relations, and axioms.

Ontologies qualify for the Enterprise SOA scenario because they serve as a unifying framework for different viewpoints and aim at reducing conceptual and terminological

confusion [UG96]. First, different stakeholders have to gain an enterprise-wide understanding of the problem domain. In our case, it includes an abstract model (conceptualization) of the concepts and relations together with their intended meaning, e.g., the meaning of "service" and "business object". Second, ontologies allow for the integration of heterogeneous information from various sources [St01]. This way, formalized knowledge can be more easily transferred and translated into different perspectives. Third, ontologies can be expressed using a knowledge representation language, which has a sound formal basis. This enables inference services and automated consistency checks, which is another requirement of our system. Recently, standards for ontology representation such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL) have emerged. RDF is a simple graph-like format for describing metadata about resources[1]. OWL[2] is defined on top of RDF(S) and provides a standard ontology vocabulary for describing ontologies based on description logics.

Due to these advantages ontologies have become the leveraging element in many knowledge management approaches [Ma03, OL98]. With the emerging vision of the Semantic Web [BHL01], ontologies have also attained increasing attention in the Software Engineering community. The potential applications in Software Engineering are manifold since ontologies can be used at development-time as well as runtime [HS06]. Our approach can be classified as ontology-enabled (cf. [HS06]) because it uses ontologies as its infrastructure for supporting development activities.

3.2 Semantic Wikis

Due to their low entry barriers and collaborative features, wikis are a lightweight approach to web-based content management, which allows multiple users to create documents on a shared subject of interest [LC01]. They have thus become a popular documentation tool in software processes (see [De05] for an overview). However, traditional wikis exhibit weaknesses when it comes to structuring the content of a wiki page. Although the set of "pages" forms a top-level structure, the underlying page content cannot be structured.

This has led to the idea of "semantic wikis". If some wiki content was structured and made machine-interpretable, a site like the Wikipedia could heavily benefit because its pages contain a lot of useful and potentially machine-processible knowledge [Vö06]. Several projects have thus proposed semantic extensions to the wiki approach. They all have in common that they allow structured knowledge to be described in a formal language, instead of processing solely hypermedia-based content. This is either done by appending metadata to wiki pages or by including knowledge inside the unstructured text by using extensions to the wiki markup language. The latter approach is used by the SemanticMediaWiki project [Vö06], which extends the existing wiki markup to enrich hyperlinks between wiki pages with semantic relations.

[1] http://www.w3.org/RDF/
[2] http://www.w3.org/TR/owl-features/

Semantic wikis interpret wiki pages as entities, and hyperlinks between wiki pages as relations among entities. Due to the additional semantic descriptions the implicit structure is made explicit, and a machine-processible knowledge model can be derived. Current semantic wikis differ concerning the creation and maintenance of this explicit kind of knowledge. While some require upfront knowledge engineering efforts, other approaches allow for continuous refinement and formalization of knowledge by the users. As we will line out in the following section, we prefer a dual approach.

Clearly, semantic wikis are a prime candidate for knowledge sharing in our case, because they provide a user-friendly way for searching and browsing structured information. Another advantage is that they combine informal with formal descriptions, thus closing the gap between the business-oriented and technical perspective on an architecture.

4 Ontobrowse Semantic Wiki

Based on wikis and ontologies as building blocks, we now describe the general architecture and prototypical implementation of our approach.

4.1 General Architecture

In the previous sections we identified the integration of architectural information from external sources as one key requirement. In Ontobrowse we thus distinguish two layers. The first layer is the artifact layer where the stakeholders maintain descriptions of services using their own tools and formats. In order to integrate the various artifacts into a mutual knowledge base, they have then to be mapped according to an ontology. Moreover, we need the semantic wiki as a user interface for presenting, querying and searching the knowledge base.

The resulting conceptual architecture of Ontobrowse is depicted in Figure 3. The integration layer includes the following components: a Web interface, a wiki manager, an ontology API to access the knowledge base, and a plugin manager.

The central part in the integration layer is the knowledge base, which is formed by one or more ontologies and instance data. It is processed using an ontology API and an underlying reasoner. While the ontologies define the knowledge structure, i.e., the boundaries in which instances can be described, instance data refers to the individual objects and their property descriptions conforming to the ontology. For example, a SOA ontology may specify the concepts “service” and “business object” together with their properties and axioms. The instances are represented by actual services and business objects developed in a SOA project. Each concept, relation, or individual is displayed by the wiki manager as a “wiki page”. It contains properties that make statements about this page, e.g., a business object which is semantically described by a domain concept. We also refer to a wiki page as an “entity”, because it is contained in the knowledge base and can be requested with a unique identifier (URI).

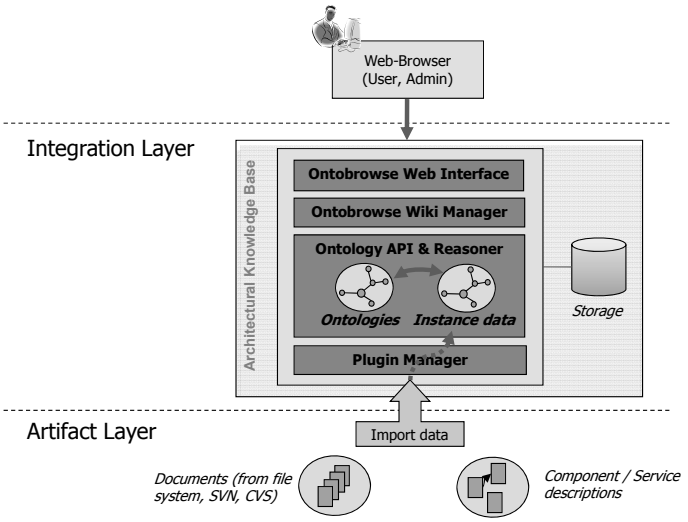

Figure 3: Ontobrowse architecture [HS07]

The wiki manager bundles the functions for fulfilling the requirements, such as processing page requests, editing textual documentation and instance property values, searching and deductive querying, and user authentication. Entity (page) descriptions are returned by an ontology API, which wraps the underlying reasoner and ontology processing tools.

Typically, ontologies are constructed upfront using an ontology editor such as Protégé[3] and uploaded by an administrator using the wiki manager. Within the knowledge structure defined by ontologies, it is possible to add instance data in two different ways: First, a wiki user can use the interface to describe properties – may it be text-based or metadata-based – about instances of concepts. Second, external tools can plug into the wiki application and map architectural description resources to instances in the knowledge base.

This leads us to the plugin manager. To a great extent, the instance data is embodied in applications and artifacts, which are managed outside the wiki, e.g., service specifications in the Enterprise SOA case. This data has to be imported from external sources, such as configuration management systems. The plugin manager allows mapping external artifacts to instance data and add this data to the knowledge base. This component exposes standard interfaces, which allow tools to retrieve artifacts, map them according to an ontology, and create or update instance data in the knowledge base. As an example, the operations of a service can be automatically extracted from service description artifacts.

[3] http://protege.stanford.edu

4.2 Design and Implementation

In this section, we will describe a concrete implementation of the conceptual architecture according to the layers of the architecture, which were introduced in the previous section.

Since SOA artifacts tend to be maintained in various heterogeneous systems and data sources, such as XML descriptions, documents, or databases, the artifact layer deals with the extraction of facts from these sources,. As a framework for this task, we use the Open Source framework Aperture[4]. Aperture comes with a number of physical connectors, such as for crawling file systems or the web, which we complemented by crawlers for SVN and CVS repositories, as well as direct connections to JDBC databases and issues tracking systems such as JIRA.

In Aperture, the objects that are crawled from those sources are directed to so-called Extractors. These extract metadata in RDF. Besides the built-in extractors for various common document formats (e.g., Microsoft Office and PDF), we added extractors for WSDL, JIRA, and Java source code.

The artifact layer is extensible in two ways: additional repositories of supported types (such as an additional SVN repository to crawl) can be easily configured by XML files. To include new data sources, Aperture provides a modular infrastructure, which just requires the implementation of two Java interfaces.

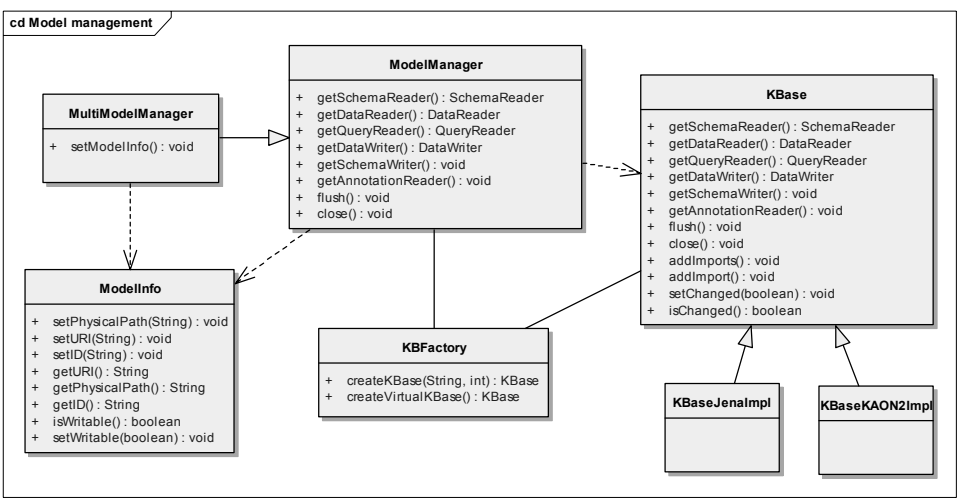

Figure 4: Ontobrowse Model Manager

The metadata extracted by the Aperture crawlers is written to the Ontobrowse knowledge base. In order to encapsulate the concrete metadata store used, we developed an abstraction API called "KOntoR API", which we briefly describe in the following.

[4] http:// aperture.sourceforge.net/

The API consists of two major parts: the model management package and the ontology API. The model management mainly consists of the ModelManager and ModelInfo classes (cf. Figure 4). ModelInfo encapsulates a file serialization of an ontology. It serves as a parameter for ModelManager for dealing with only one ontology or MultiModelManager, when dealing with a set of ontologies.

The ModelManager uses a KBFactory class to instantiate a KBase using either a KAON2 reasoner[5] or a Jena metadata store[6] as a backend. However, the concrete backend remains hidden to the using classes. After instantiation, the content of the ontologies can be accessed and modified using the different interfaces provided by ModelManager.

There are interfaces for reading certain ontology information (e.g., SchemaReader, DataReader), writing data (e.g., DataWriter), and querying (QueryReader). These interfaces again have special implementations for each backend (e.g., for KAON2 or Jena). The methods of the interfaces map to the atomic entities of the data API, which will be described in the following.

The Ontology API serves as a lightweight, partial representation of a graph structure. The subsystem consists of the following classes (see Figure 5):

KBEntity: This is the abstract base class of an entity in the knowledge base. An entity is characterized by a name (label) and a URI.

Concept: This is the representation of a concept (or "class" in OWL terms) in a knowledge base. It is a lightweight representation, since it does not include information about individuals, datatypes, or object properties.

RichConcept: This is a heavyweight representation of a concept. In contrast to the Concept class, it contains information about the datatypes and the properties.

Individual: This is a lightweight representation of an individual (also called "instance") in a knowledge base.

RichInvidual: This is a heavyweight representation of an individual. It contains datatype and object properties with values. Note that values of object properties are again (lightweight) Individuals, which can be resolved to RichIndividuals.

ObjectProperty/RichObjectProperty: The ObjectProperty classes encapsulate relations between concepts, which exist in the knowledge base.

DatatypeProperty/RichDatatypeProperty: The DatatypeProperty classes represent attributes of concepts, which are of base types such as String, numbers, or date values.

[5] http://kaon2.semanticweb.org
[6] http://jena.sourceforge.net/

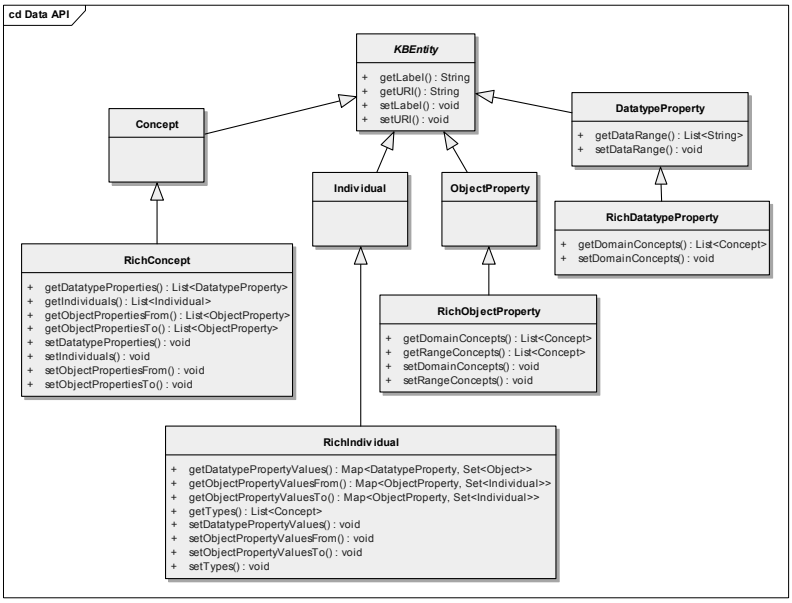

Figure 5: Ontobrowse Ontology API

Existing Java APIs for OWL knowledge bases (e.g., Protégé API[7], OWL-API[8]) are mostly based on (a) representing the whole ontology graph in memory and (b) supporting the full set of axioms for the underlying knowledge representation language.

In contrast to this, our design goals are:

- Provide a lightweight and stateless API for knowledge base access
- Focus on instance retrieval and manipulation, and omit sophisticated schema manipulation
- Provide an abstraction layer for ontology stores/engines

Thus, our API does not replace, but complements the APIs of existing ontology stores. We rely on the existing implementation e.g., for loading ontologies and executing reasoning tasks at the low level, and provide a high level representation, which abstracts from most complexities in ontology handling. Due to the abstraction layer, a further advantage of our API is that it abstracts from the specifics a concrete knowledge representation language. Our API is not necessarily limited to Semantic Web languages, since it could also have an implementation based on relational databases.

Currently, we have implementations of our API for the Jena Semantic Web framework, as well as for the KAON2 OWL reasoner.

[7] http://protege.stanford.edu/plugins/owl/api/
[8] http://owl.man.ac.uk/api.shtml

Besides the ontology API, Ontobrowse offers service interfaces for importing and updating ontologies and for managing wiki pages as well as user accounts.

While the backend services can be accessed via arbitrary clients, our standard user interface is a web application implemented with Java Server Faces[9]. Currently, this web application includes dialogs to browse the knowledge base (list concepts, view concepts and instances– see e.g., Figure 5, specify and execute SPARQL queries and for user management. We are currently extending the user interface to allow for editing property values and to add relations to link different entities in the knowledge base.

5 Setting Up Ontobrowse in a SOA Environment

In this part, we describe the necessary steps for setting up Ontobrowse in a concrete SOA environment. Initially, all stakeholders need to agree upon a shared conceptual structure, a so-called "SOA ontology" (cf. Figure 6). This ontology should capture a shared understanding of both business experts and technical people. Typically, it includes concepts like "service", "interface", "business object", and "domain concept". On the one hand, "service" defines data types and properties from the technical domain, such as "version" and "hasInterface". On the other hand, it includes properties relevant in the business view, such as "refersToDomainConcept" to reference a project glossary term. The specification has to be carried out by ontology engineers, creating an ontology file with an editor such as Protégé. An ontology file is uploaded to Ontobrowse via the Web interface and subsequently processed by the Ontology API.

[9] http://java.sun.com/javaee/javaserverfaces/

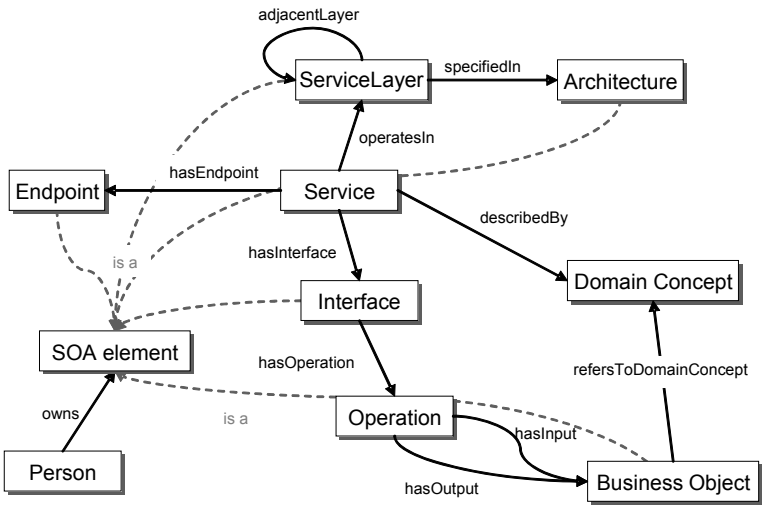

Figure 6: SOA ontology

The stakeholders can also decide whether they reuse existing ontologies. Potentially useful sources include the foundational ontologies being developed within Semantic Web Services [Ak05, ES05, OW04] and the Web Services Architecture [W3C04]. Of course, it is possible to develop several modular ontologies covering various information needs, e.g., project management and organizational structure. Instance data corresponding to the SOA ontology may be created either directly in the wiki or imported from external sources by defining plugins. This ensures high flexibility and enables to augment SOA elements with additional descriptions.

Plugins perform the actual mapping of instances from an external source into the knowledge base, e.g., from WSDL service descriptions maintained in a file system to service descriptions in the wiki. Here, we give an example how the mapping works for WSDL. However, the process is analogous for other formats, e.g., the Business Process Execution Language for service composition.

First, a one-way mapping between WSDL service descriptions and the SOA ontology is defined. In order to accommodate service properties such as version and architectural layer, we extended the WSDL format. The actual mapping is executed by a Java program, which conforms to the Ontobrowse plugin interface. It takes a WSDL file as input and performs a set of actions for adding instances, properties and attributes to the knowledge base with the Ontology API. A wiki administrator configures the input sources (CVS, file system) and update types (manual, timer task, update event). Based on this configuration the plugin manager component is responsible for updating the knowledge base automatically.

Figure 7: Concept page for "service"

Once the initial structure and wiki content has been created, it is possible to access the knowledge base through the Web interface. First, users can quickly gain an overview by starting with a concept page. For example, the page for the concept "service" shows the sub-concepts, all instances to that concept as well as the incoming and outgoing properties (cf. Figure 7). A user can then navigate to a service to read its detailed description. Second, there is a full text search of all entities in the knowledge base. Third, there is also the possibility for looking for very specific knowledge. A query interface enables users to define chained queries consisting of sentences with *subject*, *predicate* and *object* (e.g., all services "x" defining interface operations with the output "Customer"). Matching entities are returned for the variables defined by the query.

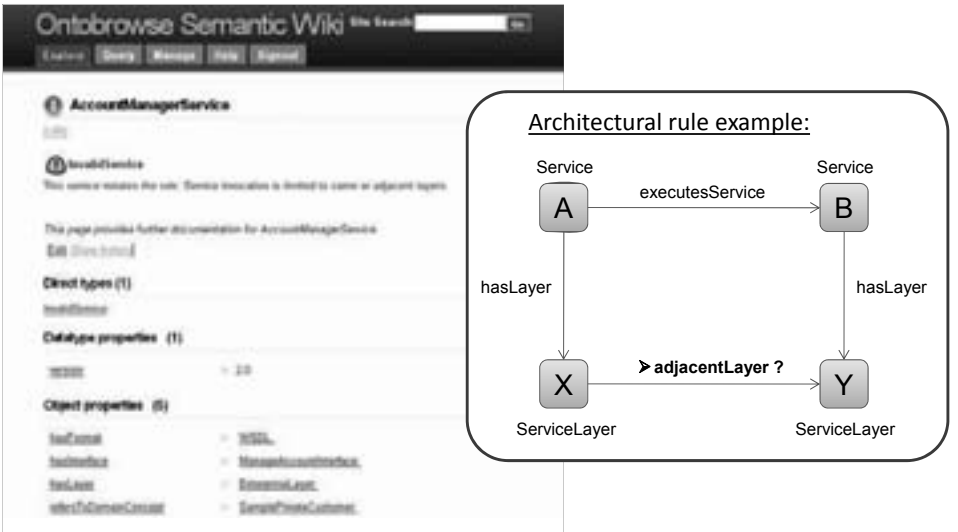

Figure 8: A service violating an architectural rule

Finally, the SOA ontology can be enhanced by rules, which enable automatic consistency checking of entities and generation of new knowledge. So far, we have included experimental support for DL-safe SWRL[10] rules in the KAON2 configuration. One application scenario is the formal definition of architectural rules, which are usually only informally documented by software architects. The semantic wiki makes the violation of these rules explicit, thus supporting their enterprise-wide enforcement. For example, we stated the rule "services may not call other services more than one layer apart" (see section 2.2) as visualized in Figure 8:

$$\text{executesService}(A, B) \wedge \text{hasLayer}(A, X) \wedge \text{hasLayer}(B,Y) \wedge \neg\, \text{adjacentLayer}(X,Y) \wedge \neg\, \text{owl:sameAs}(X,Y) \Rightarrow \text{InvalidService}(A)$$

Any entity violating this rule is categorized as an "invalid service". By using ontology annotations for the concept "invalid service" the Web interface can display a warning to the user. Alternatively, it is possible to filter for all invalid services using the query interface. To this end, Ontobrowse not only improves the navigation, documentation, querying and searching but also contributes to the quality of an Enterprise SOA.

6 Conclusion

In this paper, we described an approach based on ontologies and semantic wikis, which tackles key issues in the documentation of an Enterprise SOA. The SOA case revealed the distributed character of the SOA development process, which has been insufficiently addressed so far. Because an Enterprise SOA not only involves multiple roles, but also

[10] http://www.w3.org/Submission/SWRL/

brings different organizational units and external service providers together, the responsibilities (and with it architectural information) are inherently distributed.

Although a SOA leads to a higher degree of standardization at first glance, it nevertheless involves different views, which are either technical or business-oriented. This results in a high number of heterogeneous, locally maintained SOA artifacts with varying degrees of formalization. What is sought after is therefore both a “common language” shared by all stakeholders and a first-class representation for different types of architectural information. As pointed out in this paper, ontologies can provide the means for solving the terminological and the information integration problem. Semantic wikis on the other hand, provide a flexible way for accessing this information, e.g. browsing searching, and semantic querying. Most importantly, the proposed solution can be tailored to project-specific needs by defining one or more ontologies to set up the initial structure of the wiki.

References

[AD05] Aguiar, A.; David, G.: WikiWiki weaving heterogeneous software artifact. In: Proc. of the 2005 international symposium on Wikis, San Diego, CA, 2005, pp. 67-74.

[Ak05] Akkiraju, R.; et al.: Web Service Semantics - WSDL-S. W3C Member Submission, 2005.

[BM05] Bachmann F.; Merson, P.: Experience Using the Web-Based Tool Wiki for Architecture Documentation. Technical Note CMU/SEI-2005-TN-041. September 2005.

[BHL01] Berners-Lee, T.; Hendler J.; Lassila, O.: The Semantic Web. Scientific American. May, 2001.

[De05] Decker, B. et.al.: Self-organized Reuse of Software Engineering Knowledge supported by Semantic Wikis. In: Proc. of Workshop on Semantic Web Enabled Software Engineering, Nov. 2005.

[ES05] ESSI WSMO: Web Service Modeling Ontology (WSMO). http://www.wsmo.org/, 2005.

[Gr93] Gruber T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. 5, 1993, pp. 199-220.

[HS06] Happel, H.-J.; Seedorf, S.: Applications of Ontologies in Software Engineering. In: Proc. of Workshop on Sematic Web Enabled Software Engineering" (SWESE) on the ISWC 2006, Athens, Georgia, November 5-9, 2006.

[HS07] Happel, H.-J.; Seedorf, S.: Ontobrowse: A Semantic Wiki for Sharing Knowledge about Software Architectures. In: Proc. of the 19th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), Boston, USA, July 9-11, 2007, pp. 506-512.

[HS05] Huhns, M.H.; Singh, M.P.: Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing, vol. 9, no. 1, 2005, pp. 75-81.

[Kr95] Kruchten, P.: The 4+1 View Model of Architecture. In: IEEE Softw. 12 November, Nr. 6, 1995, pp. 42-50.

[LC01] Leuf, B., Cunningham, W.: The wiki way: Quick collaboration on the web. Addison-Wesley, 2001.

[Ma03] Maedche, A. et.al.: Ontologies for En-terprise Knowledge Management. IEEE Intelligent Systems ,18, 2003, pp. 26-33.

[MT00] Medvidovic, N.; Taylor, R. N.: A Classification and Comparison Framework for Software Architecture Description Languages. In: IEEE Trans. Software Eng. 26(1): 2000, pp. 70-93.

[OL98] O'Leary; D.E.: Using AI in Knowledge Management: Knowledge Bases and Ontologies. IEEE Intelligent Systems, 13, 1998, pp. 34-39.
[OW04] OWL Services Coalition: OWL-S Semantic Markup for Web Services. 2004.
[St01] Staab, S. et.al.: Knowledge Processes and Ontologies. IEEE Intelligent Systems, 16, 2001, pp. 26-34.
[UG96] Uschold, M.; Gruninger, M.: Ontologies: principles, methods, and applications. Knowledge Engineering Review, 11, 1996, pp. 93-155.
[Vö06] Völkel, M. et.al.: Semantic Wikipedia. In: Proceedings of the 15th International Conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006.
[W3C04] W3C: Web Services Architecture. W3C Working Group Note, 11 February, 2004.
[We03] Welty, C.A.: Software Engineering. In: Description Logic Handbook, 2003, pp. 373-387.
[WF99] Welty, C.A.; Ferrucci D.A.: A Formal Ontology for Re-Use of Software Architecture Documents. ASE, 1999, pp. 259-262.