TMLAI TRANSACTIONS ON MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE
SOCIETY FOR SCIENCE AND EDUCATION UNITED KINGDOM
VOLUME 2, ISSUE 6
ISSN: 2054 - 7390

Ontological Support for the Evolution of Future Services Oriented Architectures

[1]Bilal Gonen, [1]Xingang Fang, [1]Eman El-Sheikh, [1]Sikha Bagui, [1]Norman Wilde and [2]Alfred Zimmermann
[1]Department of Computer Science, University of West Florida, Pensacola, FL, USA;
[2] Department of Informatics, Reutlingen University, Reutlingen, Germany;
bgonen@uwf.edu; xfang@uwf.edu; eelsheikh@uwf.edu; bagui@uwf.edu; nwilde@uwf.edu;
alfred.zimmermann@reutlingen-university.de;

ABSTRACT

Services Oriented Architectures (SOA) have emerged as a useful framework for developing interoperable, large-scale systems, typically implemented using the Web Services (WS) standards. However, the maintenance and evolution of SOA systems present many challenges. SmartLife applications are intelligent user-centered systems and a special class of SOA systems that present even greater challenges for a software maintainer. Ontologies and ontological modeling can be used to support the evolution of SOA systems. This paper describes the development of a SOA evolution ontology and its use to develop an ontological model of a SOA system. The ontology is based on a standard SOA ontology. The ontological model can be used to provide semantic and visual support for software maintainers during routine maintenance tasks. We discuss a case study to illustrate this approach, as well as the strengths and limitations.

Keywords: Ontology; ontological modeling; services oriented architecture; software evolution; software maintenance; semantic support.

1 Introduction

Services Oriented Architectures (SOA) have emerged as a useful framework for developing interoperable, large-scale systems, typically implemented using the Web Services (WS) standards [1]. SOA typically refers to large systems-of-systems in which composite applications are created by orchestrating loosely coupled service components that run on different nodes and communicate via message passing [2]. Often an infrastructure layer, sometimes called an Enterprise Service Bus (ESB), mediates the communication, providing features such as routing, security, and data transformation. Such systems present several software engineering challenges because they need to orchestrate diverse services having different owners, and have complex reliability requirements. While developing SOA applications presents many software engineering challenges, managing the evolution of such systems presents even greater challenges [3][4].

SmartLife applications are emerging as intelligent user-centered systems that will shape future trends in technology and communication, including social networks, smart devices, and intelligent cars. The development of such applications integrates web services, service-oriented enterprise architectures, cloud computing and Big Data management, among other frameworks and methods [5][6]. SmartLife

DOI: 10.14738/tmlai.26.784
Publication Date: 21th December, 2014
URL: http://dx.doi.org/10.14738/tmlai.26.784

systems present even greater software evolution challenges. Such applications exist in complex and dynamic real-world environments with frequent new functional and interface requirements, rapidly emerging security issues, partner services that may be withdrawn or modified at short notice, etc. Software maintainers will need to implement changes rapidly to minimize downtime, yet they will be dealing with a complex combination of external services and internal legacy code, probably not of their own writing.

Although ontologies and ontological modeling have been used to support program comprehension, their integration and application to support maintenance and evolution of complex SOA systems, including SmartLife systems, is novel. Our research integrates multiple research directions to support the evolution of services oriented architectures and address the challenges of future SOA systems [7]. First, we extend a standard SOA ontology to develop a SOA evolution ontology that better supports typical maintenance tasks. We then use this ontology to develop a model of a SOA system and demonstrate how such ontological models can be used to provide semantic support for the maintenance of SOA systems.

In this paper, we present the development and use of ontological modeling methods to address the emerging challenges of SOA evolution. The next section describes Services Oriented Architectures and issues involved in SOA maintenance, including background and related work. Section 3 describes the use of ontologies for modeling SOA systems and specifically for maintenance of such systems. We then show how a standard SOA ontology can be extended to support maintenance in Section 4, followed by a case study that demonstrates this approach on an existing SOA application in Section 5. Finally, we show how such an ontological model can be used in a semantic browser to provide visual support for a maintainer in Section 6, followed by a discussion of conclusions and future work in Section 7.

2 SOA and SOA Maintenance

Services Oriented Architectures (SOA) is not a specific architecture, but rather an architectural style for constructing large software systems. Though definitions of SOA vary, these composite applications are constructed by orchestrating collections of services that run on different nodes and communicate by message passing. Often the messages are mediated by an Enterprise Service Bus (ESB) that provides loose coupling by handling routing, fault tolerance, data translation, etc. (See Figure 1). The services themselves may be heterogeneous, in that they may be implemented using different programming languages, operating systems and vendor infrastructure. Ideally each service represents some specific business functionality but services may also be created by wrapping existing legacy software.

To achieve interoperability, implementers of SOA often adopt some of the Web Services standards [1]. In this case each service will publish an interface in Web Services Description Language (WSDL) and the data contained in messages will be defined in XML Schema Definitions (XSD). In some cases the code will be written using Business Process Execution Language (BPEL) that is essentially a programming language encoded in XML for orchestrating interactions among services.

Large corporations have been adopting SOA because it allows them to create components that can be used across the organization with relatively little inter-divisional coordination. An example is provided by a major multinational insurance group, which has grown by acquisitions to encompass diverse

divisions each with their own IT infrastructure. To control costs and improve agility, the group is developing reusable services for common functions such as claims processing [8].

A great strength of SOA is the flexibility provided by the loose coupling, both between the services themselves and between the IT organizations that manage them. But this flexibility may create significant problems when the composite application needs to evolve. Several authors have pointed out characteristics of SOA that may make such maintenance difficult [3][4][9]. An important part of any maintenance task is acquiring the knowledge needed to make changes to the software; changes made based on incomplete understanding may fail with disastrous results. For full comprehension a maintainer may need to understand many of the technologies that the system uses, including some that may be not be common within his own branch of the organization. Even if he has the necessary technical knowledge, not all of the code or documentation may be available to him.

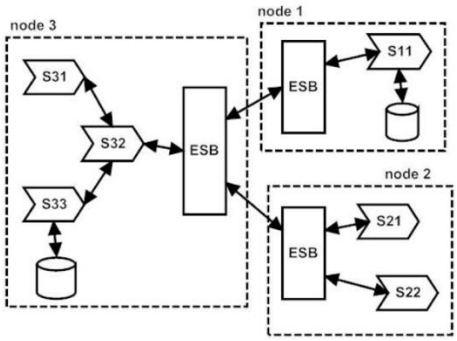

Figure 1: A SOA Composite Application

As to the evolution of future SOA systems with still weaker organizational links, problems of governance may become increasingly challenging. Changes to requirements may need to be negotiated between partners who are not accustomed to close collaboration. Security issues may arise at short notice and it may be difficult to evaluate their impact without knowing details of partners' infrastructure. The mix of partners may also change quickly as new services are offered and old ones withdrawn.

There has been a moderate amount of research on tools and methods to help the maintainer of a SOA application. Papazoglou, Andrikopoulos, and Benbernou categorize changes into "deep" and "shallow" and describe how to perform a "gap analysis" between the existing software and the desired improvement [10]. To provide a view of how an application operates across multiple nodes, several authors propose using dynamic analysis, generally starting from some form of log of inter-process messages. One early proposal was IBM's Web Services Navigator, which provides several visualizations of message logs [11]. A later paper from the same group identifies correlations between different messages by looking at values in specific data fields [12]. An interesting problem sometimes addressed by dynamic analysis is the location of the code that implements one specific "feature" or functionality of a composite application. One technique computes a component relevance index for each message and then displays the messages using a Feature Sequence Viewer [13]. In another approach, Yousefi and Sartipi propose identifying features using an analysis of dynamic call trees from distributed execution traces [14].

Dynamic analysis techniques are very powerful, but it can be difficult to collect the necessary data since usually a crafted set of tests needs to be run on a system which is instrumented to allow the collection

of correlated message logs from the different nodes. Static techniques instead rely only on interface descriptions, documentation, and possibly some source code. However they often cannot give as complete results without considerable human input. For example one method recovers concept maps from the interface descriptions as a starting point for knowledge engineering interviews with system experts [15]. Another approach uses search methods, enhanced by a rule-based system for extracting abstractions from the service interfaces [16].

3 Ontological Modeling

In philosophy the concept of ontology refers to the study of things that exist [17][18]. Ontologies define content theories and representations of concepts as objects, their properties and their relationships. Ontologies specify terms for describing specific domain knowledge by representing domain facts using a structured vocabulary of concepts [17]. The body of knowledge describes a specific domain using that vocabulary as a collection of facts. One most common definition of the term "ontology – as a specification of a conceptualization" is from [19] and [20]. Current mechanisms for representing ontologies are closely related and seem to be similar to the object-oriented terminology. A comparison of ontology terms and meanings in different contexts is in [21]. The ontological terms have object-oriented correspondences with similar meanings: concept relates to class, individual to instance, property of object to relationship, property of data-type to attribute, schema or ontology to class model, and knowledge base to object model.

In our understanding architecture evolution ontologies represent a common vocabulary for software engineers [21] who need to share their information based on explicitly defined concepts. Ontologies include the ability to infer automatically transitive knowledge. Our ontological approach has some practical reasons: share the common understanding of service-oriented architecture domains and their structures; reuse the architectural knowledge; make architectural requirements, structures, building blocks explicit; promote reusability of architectural artifacts; separate the architectural knowledge according to orthogonal architectural domains; and classify, analyze, and diagnose enterprise systems according to the service-oriented reference architecture.

For our purpose, as in [17], [18], the ontology is a formal and explicit description of shareable and automatically navigable concepts of our architectural domain. For modeling purposes we use UML class diagrams to represent concepts, and we describe the attributes as properties (sometimes called roles) and role or property restrictions as facets. This ontology structure, together with the instances of these concepts, constitutes the knowledge base. Practically the knowledge base is a growing structure, which starts with the basic concept structures and is enlarged by more or less an amount of growing number of instances.

We have developed exemplarily metamodels and related ontologies seeded by a student research project [22] for the following main architectural domains from the Enterprise Services Architecture Reference Cube (ESARC) as a starting and extendable set of work results: Business & Information Reference Architecture, Information Systems Reference Architecture, and the Technology Reference Architecture. Metamodels are used, as standardized in [23], to define architecture model elements and their relationships for the reference architectures of ESARC. Metamodels define models of models. In

our approach for architectural modeling we use metamodels as an abstraction for architectural elements and relate them to architecture ontologies.

The Reference Model for Service Oriented Architecture of OASIS [24] is an abstract framework, which defines basic generic elements and their relationships of a service-oriented architecture. This reference model is not a standard, but provides a common semantic for different specialized implementations. Reference models are, as in [24], abstract conceptual models of a functional decomposition of model elements together with the data flows between them. Reference architectures, in [25] and [26], are specialized models of a reference model. It is a composition of related architectural elements, which are built from typed building blocks as the result of a pattern-based mapping of reference models to software elements.

The technical standard of Service Oriented Architecture Ontology from [27] defines core concepts, terminology, and semantics of a service-oriented architecture in order to improve the alignment between the business and IT communities. Following stakeholders are potential users of the SOA ontology, related architecture metamodels, as well as concrete architectural building blocks: business people and business architects, architects for the information systems and software architecture, architects for the technological infrastructure, cloud services architects and security architects.

The SOA Ontology in [27] is represented in the Web Ontology Language (OWL) [18]. The ontology models the core concepts of SOA as classes and properties. The SOA ontology includes additional natural language description of the main concepts and relationships in UML diagrams, which show graphically the semantic concepts as classes and the properties as UML associations. The intent of the UML diagrams are for explanations only, but are helpful constructs for understanding the modeled domain of SOA architecture and more concise than the more spacious formal descriptions in OWL. The SOA ontology defines the relationships between semantic concepts, without mentioning the exact usage of these architectural concepts. To illustrate the SOA ontology the standard uses examples and descriptions of these in natural language.

The two core concepts of the SOA ontology, as in [27], are: System and Element. These two core concepts are generic and often used concepts to define a composite structure of systems that have elements. These abstract meaning of systems and elements is used in different specific architectural modeling situations. Using the concept "*Element*" the technical standard associates following core properties: *uses* and *usedBy* as well as the properties *representedBy* and *represents*. The technical standard of SOA Ontology defines additional concepts of the SOA Ontology like *HumanActor*, *Task*, *Service*, *ServiceContract*, *Effect*, *ServiceInterface*, *InformationType*, *Composition*, *ServiceComposition*, *Process*, *Policy*, and *Event*.

4 A SOA Evolution Ontology

The starting point for our SOA Evolution Ontology is the Open Group SOA ontology technical standard [27], developed by the Open Group in order to provide a common understanding of services oriented architectures and help improve alignment between the business and information technology community. Compared to other service-related ontologies such as OWL-S, WSDL-S and WSMO that focus mostly on the application of service discovery and invocation and composition, the Open Group SOA ontology seems a more appropriate point of departure given the maintainer's need to comprehend the system at multiple levels. The Open Group Ontology is defined in the web ontology language (OWL)

and is ready for extension and population for specific applications. In this ontology, 15 classes and 30 object properties are defined. The class hierarchy is shown in Figure 2.

To develop the SOA Evolution Ontology, the Open Group SOA ontology was extended to improve support for software maintainers. Specifically, the Service class was sub-classed into InternalService and ExternalService classes to distinguish services owned by the maintainer's organization from those owned by partners. A maintainer would have much greater information and control for the internal services. A ProcessingModule class was added as a subclass of Element to model code components that perform the InternalServices. The software artifacts that perform ExternalServices are out of the SOA system boundary. A subclass of the top-level Thing class DataItem was added to contain information on every data item appearing in service input or response messages (InformationType in SOA ontology). Detailed data items would add a lot of information to help a maintainer trace problems located in message transmission. The extended class hierarchy is shown in Figure 3. The corresponding object properties hasDataItem and isDataItemOf were defined between InformationType and DataItem classes.

Figure 2: The Open Group SOA Ontology Class Hierarchy

Figure 2: The Extended Open Group SOA Ontology Class Hierarchy

These extensions enhance the ability of the ontology to support software maintenance queries. With the added classes, much more relevant information for maintenance purposes may be stored as individuals (the OWL term for class instances) in the ontology and thus be available for future queries.

In a typical populated ontology for a SOA system, the basic building blocks are service clusters with InternalService or ExternalService individual in the centre with ServiceInterface, InformationType and DataItem individuals attached directly or indirectly. A service in SOA ontology is different from a service in a SOA system. In a SOA system a service (as defined in a single WSDL file) may consist of multiple operations while a service in SOA ontology only represents a single operation to simplify the relationship between service class (InternalService or ExternalService) individuals and ServiceInterface class individuals. With service defined as a single operation, only one ServiceInterface individual is attached to a service, which reduces the complexity and confusion. Thus, each InternalService or ExternalService individual has a ServiceInterface individual attached to define its interface and each ServiceInterface has two InformationType individuals correspond to its input and output messages. Each InformationType individual contains multiple DataItem individuals that represent fields in a message. Service clusters are

connected by System and ProcessingModule individuals. A System may use multiple services while a ProcessingModule may perform an InternalService and use multiple services. In this way, all service clusters are connected to form the complete SOA ontology.

Due to the complexity of a SOA system, the number of instantiated individuals in the extended SOA ontology is expected to be beyond the capability of manual population. Fortunately, the design of an ontology is capable of accommodating automatic or semi-automatic populations of ontologies, depending on the programming language that the SOA system is coded in. In a typical SOA system, most of information need for the ontology population is defined in standard WSDL, XSD, and optional BPEL files. Everything in a service cluster can be found in a single WSDL file and its related XSD files. With these, Services (belonging to InternalService or ExternalService classes) are instantiated to generate ontology individuals with their related ServiceInterface, InformationType and DataItem individuals. The individuals of these five classes can be populated from the WSDL and XSD files automatically or semi-automatically. Individuals belonging to these five classes consist of the biggest portion of the whole populated ontology. The rest, which is a small portion of individuals, belong to the System and ProcessingModule classes, will be populated to connect the service clusters. Here, the automatic population is possible under specific circumstances (for example, a system all written in BPEL), but it will not be easy because of the relationships that exist in programs written in different programming languages. So the invocation and dependence of relationships between services have to be manually extracted from source code to complete the ontology population. Fortunately, it is likely to be only a small portion of the total number of individuals and relationships.

5 Case Study

To illustrate the use of the SOA Evolution Ontology, we populated it for a small publicly available SOA composite application called WebAutoParts that has been used in earlier studies [17]. WebAutoParts is a hypothetical online automobile parts startup that uses an agile development strategy. Instead of writing large amounts of specialized code, commercial SOA services from well-known vendors are orchestrated using a relatively small amount of in-house BPEL. This application provides an order processing workflow in which incoming orders are first checked to confirm that inventory is available, then sales tax and shipping are computed, and finally the order is stored and a note placed in a message queue to trigger order fulfillment (packing and shipping).

Vendor services are represented by their real WSDLs and include Amazon's SimpleDB and SQS simple queue services, a state sales tax service from strikeiron.com, and a shipping cost service from ecocoma.com. In-house services have XSDs for data types and WSDLs that were generated from BPEL code. The BPEL itself is stubbed since runnable code would require setting up paying accounts with each of the vendors. Table 1 lists the sizes of the artifacts that describe WebAutoParts.

Table 1: Artifacts for WebAutoParts

File	Lines
InventoryRepository.bpel	71
OrderProcessing.bpel	118
InventoryRepositoryArtifacts.wsdl	77
OrderProcessingArtifacts.wsdl	69
AmazonSimpleDB.wsdl	611
QueueService.wsdl	1043
TaxDataBasic5.wsdl	436
usps.wsdl	197
InventoryQuery.xsd	28
PurchaseOrder.xsd	36
Total	2686

Most classes of the SOA Maintenance Ontology could be populated automatically by parsing the BPEL, WSDL and XSD artifacts. However for this small test system the individuals were generated by hand, with the help of a few special-purpose scripts. The resulting individuals are as shown in Table 2.

Table 2: Populated SOA Maintenance Ontology

Class	Count
System	1
Internal Service	2
External Service	5
Processing Module	2
Information Type	14
Data Item	105
Total	129

One way to think about the contribution of the ontology is that it easily provides the maintainer with deep knowledge about the individuals in the system, as opposed to the shallow knowledge provided by simple textual search. For example, while the maintainer could use the find command in an editor to locate an XML tag containing tax Rate in an XSD, it would be much more informative to locate the corresponding individual in the ontology. The class of the individual immediately tells the maintainer he is dealing with a DataItem, and simple transitive queries can show him what Information Types it belongs to, the interfaces that use them, and the services that have those interfaces.

For each class in the ontology we defined a small packet of deep knowledge that we think would be of most use to display to a maintainer. We also defined a parameterized query to collect that deep knowledge (see Table 3).

Table 3: Suggested deep knowledge for each class in the ontology

Class of individual	Desired knowledge
InternalService	What ProcessingModule performs this service? What are this service's input and output Information Types? What other services does its ProcessingModule use?
ExternalService	What are this service's input and output Information Types? What Processing Module use this service and what are the corresponding Internal Services?
Service Interface	What services is this an interface for? What Information Types and Data Items are in its input and output?
InformationType	In what Service Interfaces does this InformationType appear as input or output? What services use those Service Interfaces? What Data Items are part of this InformationType?
DataItem	What Information Types contain this DataItem? What Service Interfaces use those Information Types? What services use those Service Interfaces?
ProcessingModule	What Internal Services does this ProcessingModule perform and what are their Service Interfaces? What services does this ProcessingModule use and what are their Service Interfaces?

To see how the ontology and the deep knowledge queries would be useful, consider three plausible maintenance scenarios for WebAutoParts:

Scenario A: Some errors have been encountered in the shipping costs computed when WebAutoParts processes orders. A Software Engineer is trying to understand how WebAutoParts gets shipping costs and what data is exchanged as part of that operation. Thus he needs to locate where the "shipping" concept is located in the system. Correct conclusion: Shipping costs are calculated by external service GetUSPSRate. If there are errors, this would be the service to check.

Scenario B: WebAutoParts has been dealing exclusively within the United States but now plans to enter other countries. One of the many changes requires understanding how the application uses American zip codes since Software Engineers will now have to modify the software to handle other kinds of postal code. The Software Engineer needs to locate where the "zip code" concept is located in the system. Correct conclusion: WebAutoParts will need to make changes to the way it computes shipping costs (currently done by external service GetUSPSRate) and to how it calculates local taxes (currently done by external service GetTaxRateUS).

Scenario C: Amazon is retiring their SimpleDB and replacing it with a new database offering. The Software Engineer needs to understand how the SimpleDB services are used in WebAutoParts. Correct conclusion: SimpleDB is used in two places. Its GetAttributes service is used by the InventoryRespository.bpel processing module while its PutAttributes service is used by the OrderProcessing.bpel processing module.

For each scenario the relevant individuals in the ontology can be found by a simple text search on "shipping", "zip" or "SimpleDB" (See Table 4). Then the packaged queries can be applied to each individual and the correct conclusions can be reached immediately.

For comparison, we asked a Software Engineer with Web Services experience to check how the same conclusions could be reached by a sequence of conventional searches on the WebAutoParts artifacts. For each search the Software Engineer used either a Unix "grep" search when he had to scan multiple files, an editor's search command when he had the problem narrowed down to a single file, or he scrolled multiple pages in a file when he needed to understand the nesting of XML elements. The starting points were the same text searches as used on the ontology. The Software Engineer tried to find

the shortest sequence of searches that would rigorously arrive at a correct conclusion. For each search he counted the number of places in the artifacts that matched his query and categorized them as:

- Useful, if they either provided him with the information he wanted or with a good starting point for the next search, or
- Non-useful, if they had to be examined and discarded.

Table 4: Comparison of Ontology and Conventional Searches

	Ontology Search	---	Conventional Search	---
Scenario	Relevant Ontology Individuals (one ontology query on each)	Searches Needed	Useful Matches	Non-useful Matches
A	DataItem: 1- USPS.GetUSPSRate.Out.Shipping	10	12	39
B	DataItems: 1- USPS.GetUSPSRate.In.ZipOrigination 2- USPS.GetUSPSRate.In.ZipDestinatio 3- TaxDataBasic5.GetTaxRateUS.Out.ZIPCode	25	36	143
C	External Services: 1- AmazonSimpleDB.PutAttribute 2- AmazonSimpleDB.GetAttributes	8	9	6

The results of the comparison are shown in Table 4. As can be seen, using the ontology drastically reduces the amount of work needed to arrive at correct conclusions about the software.

6 Ontology-Based Visual Interface for SOA Maintenance

The Semantic Web enables machines to comprehend semantic documents and data, not human speech and writings [28]. The Semantic Web provides an infrastructure that enables not just web pages, but databases, services, programs, sensors, personal devices, and even household appliances to both consume and produce data on the web [29]. Knowledge is represented in ontologies as triples. Triples consist of Subject, Predicate, and Object. For instance, John Smith likes Florida. “Likes” is a relationship between the subject John Smith, and the object Florida.

There are 267 triples in our ontology. Some of them are:

AmazonSimpleDB.GetAttributes.Interface – isInterfaceOf - AmazonSimpleDB.GetAttributes.

AmazonSimpleDB.PutAttributes - hasInterface - AmazonSimpleDB.PutAttributes.Interface.

We developed a tool “Semantic Browser” that uses ontologies for annotating documents with semantic information. The semantic browser tool has three parts: document indexing, annotation, and traversing documents. For the document indexing, we use some Java classes, i.e., Map, HashMap, HashSet classes in the Java API. We put all the documents we want to index in a folder. We select this folder from the Semantic Browser tool. We index these documents with the keywords retrieved from the ontology. There are 114 individuals in the ontology, e.g., “AmazonSimpleDB.PutAttributes”, “AmazonSimpleDB.PutAttributes.Interface”, “USPS.GetUSPSRate.In.ZipDestination”. These individuals are used to index the documents. After the documents are indexed, for a given keyword, the Semantic Browser tool returns the list of documents in which the keyword occurs.

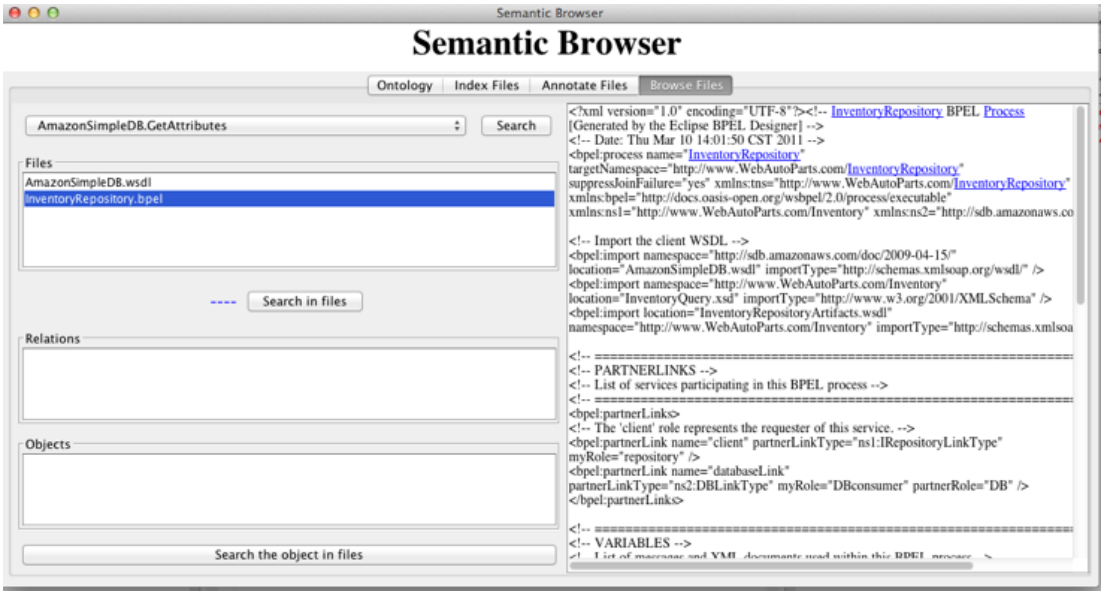

Figure 4: A Screenshot of Semantic Browser Tool

The second part of this Semantic Browser tool does the annotation automatically by processing all of the files in a folder and annotating them. The tool then looks for the individuals from the ontology in the content of the document. These terms are tagged with an html tag. For instance, the term "InventoryRepository" in the document becomes:

InventoryRepository.

The third part of the Semantic Browser tool does traversing. Figure 4 shows the traversing component of the tool. It allows users to search for related information based on knowledge captured by the ontology. Let's say as a Software Engineer, you are looking for documents containing "GetUSPSRate" and what interfaces it has.

You do an initial query on "GetUSPSRate" and are offered several files containing the term "GetUSPSRate" in them. Assume the "OrderProcessing.bpel" file contains "GetUSPSRate", and the user selects this file. The content of the file appears in the Semantic Browser. The named entities, which we have in our ontology, appear highlighted and underlined. You click on the "GetUSPSRate" in the text, and are offered some relationships, such as: "is a", "has interface". You select "has interface" relationship from the list, and are offered a list of interfaces, which come from the ontology. You select "USPS.GetUSPSRate.Interface" from the list, and are offered all of the files which contain the term "USPS.GetUSPSRate.Interface". After clicking one of the file names from the list, the content of the file appears in the browser.

Consider Scenario C explained in Section 5 let's see how we reach the same conclusion by using the ontological model. By a simple text search on "SimpleDB", we find the relevant individuals in the ontology. We start with the "AmazonSimpleDB.GetAttributes". By querying the ontology, we retrieve two triples from the ontology, which have the "AmazonSimpleDB.GetAttributes" as the subject of the triples. These two triples are:

AmazonSimpleDB.GetAttributes -> usedBy -> InventoryRepository.Module

AmazonSimpleDB.GetAttributes -> hasInterface -> AmazonSimpleDB.GetAttributes.Interface

We make another query at the document indexing part of the Semantic Browser tool to find in which document the "InventoryRepository.Module" occurs. As a result, we find that the "InventoryRepository.Module" occurs in the InventoryRepository.bpel file.

One of the benefits of the Semantic Browser tool is to provide semantic links as opposed to physical hyperlinks. Although there was no physical links between the documents, after processing the documents, the documents become linked through the semantic relationships that come from the ontology.

7 Conclusions

In conclusion, maintenance of SOA systems presents many challenges. In this paper, we argued that an ontology-based approach could ease the difficulties of maintenance and introduced an extended SOA Ontology that can be populated with information about any specific SOA-based system. The extensions provide the maintainer with more knowledge of the system, hence more control over the services, traceable steps, and the ability to support software maintenance queries. This ontology extends the Open Group SOA Ontology so it should be compatible with other tools based on this standard. As a first application of the SOA Evolution Ontology we proposed a Semantic Browser tool to aid a maintainer in navigating the many artifacts that describe a SOA system. We illustrated the approach by populating the extended SOA Ontology to model WebAutoParts, an example SOA system.

ACKNOWLEDGMENTS

Work described in this paper was partially supported by the University of West Florida Foundation under the Nystul Eminent Scholar Endowment, and the SOA Innovation Lab Germany.

REFERENCES

[1]. Nicolai Josuttis, SOA in Practice: The Art of Distributed System Design, O'Reilly, 2007, ISBN: 0-596-52955-4.

[2]. G. Lewis, E. Morris, S. Simanta and D. Smith, "Service Orientation and Systems of Systems," *IEEE Software*, Vol. 28, No. 1, 2011, pp. 58-63. doi:10.1109/MS.2011.15

[3]. Nicolas Gold, Claire Knight, Andrew Mohan, Malcolm Munro, "Understanding service-oriented software", IEEE Software, Vol. 21, March/April 2004, pp. 71-77, doi: 10.1109/ms.2004.1270766.

[4]. Grace Lewis and Dennis Smith, "Service-Oriented Architecture and its implications for software maintenance and evolution", Frontiers of Software Maintenance, 2008. FoSM 2008, pp. 1-10, doi: 10.1109/fosm.2008.4659243.

[5]. El-Sheikh, E., Bagui, S., Firesmith, D. G., Petrov, I., Wilde, N., Zimmermann, A., 2013. Towards Semantic-Supported SmartLife System Architectures for Big Data Services in the Cloud,

Proceedings of the Fifth International Conference on Advanced Service Computing (Service Computation 2013), May 27 – June 1, Valencia, Spain, pp. 59-64.

[6]. Zimmermann, A., Pretz, M., Zimmermann, G., Firesmith, D. G., Petrov, I., El-Sheikh, E., 2013. Towards Service-oriented Enterprise Architectures for Big Data Applications in the Cloud, *17th IEEE International EDOC Conference (EDOCW 2013): The Enterprise Computing Conference with SoEA4EE*, 9-13 September 2013, *Vancouver, BC, Canada*, pp. 130-135, 2013.

[7]. G. Lewis, "Is SOA Being Pushed Beyond its Limits?" Advances in Computer Science: an International Journal, Vol. 2, Issue 1, No. 2, 2013.

[8]. Deb Ayres, Dave Berry, Babic Hosseinzadeh, Conference Presentation, Oracle OpenWorld, Topic: "Service lifecycle governance", San Francisco, Sept. 21, 2010.

[9]. Nicolas Gold and Keith Bennett, "Program comprehension for web services, International Conference on Program Comprehension, 2004, doi: 10.1109/wpc.2004.1311057.

[10]. M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou, "Managing Evolving Services," IEEE Software, Vol. 28, No. 3, May/June 2011, pp. 49-55, doi: 10.1109/MS.2011.26.

[11]. W. De Pauw, et al., "Web services navigator: visualizing the execution of web services", IBM Systems Journal, Vol. 44, No. 4, Oct. 2005, pp. 821-845, doi: 10.1147/sj.444.0821.

[12]. W. De Pauw, R. Hoch, and Y. Huang, "Discovering Conversations in Web Services Using Semantic Correlation Analysis", IEEE 20th International Conference on Web Services, ICWS'2007, July 2007, pp. 639-646, doi: 10.1109/ICWS.2007.200.

[13]. John Coffey, Laura White, Norman Wilde, Sharon Simmons, "Locating Software Features in a SOA Composite Application," ECOWS, pp.99-106, 2010 Eighth IEEE European Conference on Web Services, 2010.

[14]. A. Yousefi and K. Sartipi, "Identifying distributed features in SOA by mining dynamic call trees", IEEE International Conference on Software Maintenance (ICSM), Sept. 2011, pp. 73-82, doi: 10.1109/ICSM.2011.6080774.

[15]. J. Coffey, T. Reichherzer, B. Owsnick-Klewe, and N. Wilde, "Automated Concept Map Generation from Service-Oriented Architecture Artifacts", Proc. of the Fifth Int. Conference on Concept Mapping CMC2012, Sept. 2012, pp. 49-56.

[16]. Eman El-Sheikh, Thomas Reichherzer, Laura White, Norman Wilde, John Coffey, Sikha Bagui, George Goehring, Arthur Baskin, Towards Enhanced Program Comprehension for Service Oriented Architecture (SOA) Systems, Journal of Software Engineering and Applications, Volume 6, Number 9, September 2013, pp. 435-445, doi: http://dx.doi.org/10.4236/jsea.2013.69054.

[17]. B. Chandrasekaran, J. R. Josephson, V. R. Benjamins, "What Are Ontologies, and Why Do We Need Them?", IEEE Intelligent Systems, pp. 20-26, 1999.

[18]. D. Gasevic, D. Djuric, V. Devedzic, Model Driven Engineering and Ontology Development, Springer 2009.

[19]. T. R. Gruber, “A translation approach to portable ontology specifications,” Knowledge Acquisition, Vol. 5, no. 2, pp. 199-220, 1993.

[20]. S. Staab, R. Studer (eds.), Handbook on Ontologies, Springer, 2004.

[21]. K. Schneider, Experience and Knowledge Management in Software Engineering, Springer 2009.

[22]. A. Zimmermann, G. Zimmermann, “Enterprise Architecture Ontology for Services Computing, “Service Computation 2012 Nice, France, pp. 64-69, 2012.

[23]. OMG, “Meta Object Facility (MOF) Core Specification”, Version 2.0, Object Management Group, 2006.

[24]. C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz, OASIS “Reference Model for Service Oriented Architecture” 1.0, OASIS Standard, 12 October 2006.

[25]. J. A. Estefan, K. Laskey, F. G. McCabe, and D. Thornton, OASIS “Reference Architecture for Service Oriented Architecture” Version 1.0, OASIS Public Review Draft 1, 23 April, 2008.

[26]. The Open Group, “SOA Reference Architecture”, Technical Standard, 2011, https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?catalogno=c119

[27]. The Open Group, “Service-Oriented Architecture Ontology”, Technical Standard, 2010, https://www2.opengroup.org/ogsys/jsp/publications/PublicationDetails.jsp?catalogno=c104

[28]. Tim Berners-Lee, James Hendler and Ora Lassila, "The Semantic Web", Scientific American, May 2001, p. 29-37.

[29]. James Hendler, Tim Berners-Lee and Eric Miller, 'Integrating Applications on the Semantic Web', Journal of the Institute of Electrical Engineers of Japan, Vol 122(10), October, 2002, p. 676-680