SEMF - Service Evolution Management Framework

Martin Treiber, Hong-Linh Truong, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
{treiber,truong,dustdar}@infosys.tuwien.ac.at

Abstract

With the growing popularity of Web services, an increasing number of Web services have been integrated into and used by complex service oriented systems. As a result, the management of Web services has gained more importance as Web services management systems can provide various useful, runtime and historical, service information to service consumers, developers and providers. However, current Web service management systems do not provide a holistic view of Web services. These management systems use independent information models covering different aspects of Web services, for instance, QoS, licensing, taxonomy information, to name just a few. In this paper, we address the challenges of (1) integrating available information into a common Web service information model, while (2) providing an extensible information model, and at the same time, (3) keeping track of evolutionary changes of Web services and (4) offering the means for complex analysis of Web services. We introduce a Service Evolution Management Framework (SEMF) that addresses the aforementioned challenges using a generic Web service information model. We illustrate how we utilize our proposed Web service information model to manage changes of Web services, and present a case study that shows how our framework could be used in practice.

1 Introduction

Web services were designed to address the general problem of the integration of heterogeneous applications, regardless of their implementation platforms. During the last years, Web services-based systems became more complex and management issues are becoming more and more important. There is a need to manage individual Web services and complex systems of Web services over the course of their lifetime. Being able to manage Web services helps to answer many questions related to the development and deployment of Web services, such as (i) when and based on which information a Web service should be improved, (ii) which factors influence on the employment of a Web service, and (iii) why a Web service is not widely used. Such questions are frequently asked by not only the service developer but also the provider and the consumer: they want to understand *how Web services evolve* in order to optimize the development, deployment and employment.

To understand the evolution of Web services, we need to rely on manageable information of Web services. Existing approaches provide only a fraction of information associated with Web services such as, WSOL [19], SLA [13, 18, 14], licensing information [10]. In fact, existing approaches mainly focus on interface descriptions and assume that interface descriptions can be augmented with additional information, e.g., semantic meta information [10]. However, there are diverse types of information that originate from various sources. Such diverse types of information are required by different stake-holders with different perspectives on Web services, such as developers, integrators, consumers and providers [2]. Furthermore, there is a lack of tools and frameworks that support managing and integrating of a vast source of information into a common information model.

We tackle the above-mentioned challenge by developing a holistic view on Web services related information. In particular, we address two main issues related to Web services: (i) what type of information is required for a given perspective and how to integrate all types of information into a single model, and (ii) how to collect, manage and provide those types of information. The first issue is related to the development of a novel aggregated information model for Web services. To address this, we need to ensure the flexibility and extensibility of the model and to preserve existing information models and seamlessly integrate them into the new model. The latter is related to the development of a management framework that deals with various types of information whose management strategies are different. For example, a particular type of information may change asynchronously, such as QoS parameters; one type might not be frequently changed, such as WSDL-based interface and license, while others might change rapidly such as SLA. Furthermore, Web service related information should be kept

and managed over the time because historical data provides key insights into the understanding of why Web services change and which factors impact on the changes. Such historical data can be utilized in various purposes in the selection, execution, maintenance, etc., of Web services.

This paper contributes (i) a novel, common Web service information model that integrates heterogeneous, diverse types of information about Web services, and (ii) the design and implementation of SEMF (Service Evolution Management Framework), a distributed Web services management framework that is capable of tracking changes of Web services and providing various types of Web services related information according to different perspectives.

The rest of this paper is organized as follows. Related work is discussed in Section 2. Section 3 introduces our Web service information model along with the associated roles and perspectives on the model. Section 4 describes SEMF architecture. We discuss the implementation of the current prototype of SEMF in Section 5. A case study illustrating SEMF is presented in Section 6. We summarize the paper and give an outlook for further research directions in Section 7.

2 Related work

Our work in this paper aims at (i) providing a novel information model that is capable of capturing and aggregating different types of information associated with Web services, and (ii) a framework that collects, manages and provides Web services information according to that information model. We outline our related work with respect to the information model for Web services and middleware for managing Web services.

2.1 Information Model

Category	Language
Interface	WSDL, OWL-S, WSMO, WSDL-S
QoS	ORDL-S, WSLA
Pre Conditions	OWL-S, WSMO
Post Conditions	OWL-S, WSMO
Interaction Patterns	WSMO
SLA	WSLA, WS-Policy
Taxonomy	OWL-S, WSMO
Folksonomy	-
License	ORDL-S

Table 1. Overview of Web service description languages

Existing Web service models cover different aspects of Web services such as service interface, QoS[17], SLA[13, 18, 14], licensing information [10], etc. However, those models focus on only discovery issues. Table 1 gives an overview of existing and adopted approaches and their descriptive capabilities. For example, WSDL-S [1] embeds semantic information into WSDL [5] files. The OWL-S [20] semantically models Web services from three different perspectives, namely the interface, process model and the details about the transport protocol. The WSMO framework [7] provides an ontology based solution to model Web services. The authors use a dedicated language [9] to represent semantic information about Web services. Various languages, such as WSLA [13], WSOL [19] [18], Slang [14], and tools are developed for specifying service level agreements, but they do not combine SLA-based information with other types of information, such as interface descriptions or licensing information.

The main difference of our work to the aforementioned approaches is that we focus on the management of Web services and do not assume that Web services information will be described by a single specification. We explicitly consider different types and providers of Web services information and utilize aggregation techniques to gather different sources of information which might have different representation models, such as OWL, WSDL, etc., for management purpose. Both semantic and non-semantic based models are considered in our framework.

What has not yet been provided by these proposed standards is the ability to tie together, at the point of service offering, these various sources of information in a manner which is both simple to create and use. Hence, our work aims to tackle this issue. The WS-Inspection (WSIL) specification [12] also addresses this need by defining an XML grammar which facilitates the aggregation of references to different types of service description documents, and then provides a well defined pattern of usage for instances of this grammar. In contrast to WSIL, we explicitly focus on management related information, for instance run time information about Web services (usage statistics, logging, etc.).

Recently, WS-RC (Web Service Resource Catalog) schema [11] was introduced to describe management information of IT resources. In contrast to WS-RS which is a general purpose model to describe all types of IT resources, we focus explicitly on Web services. We provide a lightweight approach for the integration of arbitrary data. Furthermore, we use existing tools to integrate distributed instances of our framework. We also provide the possibility to access the information with standard tools like email clients and Web browsers. This allows for greater flexibility, since users can for instance register for a feed that provides information about QoS changes of a Web service without having to install a dedicated client. WS-RC elements represent single entities, whereas our information model considers multiple elements for a single Web service.

2.2 Web Services Management Frameworks

Although UDDI [6] is designed to store different kinds of Web services information, it has never been widely adopted. In addition, there is no support for the integration of many types of Web services information into a common information model.

Casati et. al. focus on the business perspective of Web service management [3]. The authors provide a high level analysis of the main issues ("holistic" Service model, Metric and Architecture). They distinguish between infrastructure-level, application-level and business-level management. However, to the best of our knowledge, they do not deal with other perspectives. Furthermore, no framework has been developed to provide management aspects given in [3].

The Web Services Management Framework (WSMF) [4] defines a generic architecture for the management of resources, including Web services. WSMF focuses on managing relationships between different resources and defining monitoring interfaces associated with Web services. Similarly, WSDM (Web Services Distributed Management) [16] provides means to manage arbitrary resources and offer a set of standardized management interfaces, such as obtaining and controlling service capabilities. In contrast, we concentrate on management information associated with Web services and their evolution.

The work in [8] discusses the management of service interface change. It defines version-aware service descriptions and directory models. Our work covers multiple aspects in service management rather than only versioning. In this sense that work can be considered as a part of our approach. The proposed model in [8] can be incorporated into our management model.

3 Web Services Information Model

This section discusses our proposed Web service information model. The basic principle is to create a meta model that supports the management of evolutionary changes of Web services and integrates different sources of data into a common Web service information model.

3.1 Factors of Influence

Factors of influence can either change a Web service directly (e.g., change the interface) or have indirect effects on Web services (e.g.,QoS changes can lead to a new implementation of a Web service). Figure 3.1 depicts the four major factors of influence. These factors concern (1) the Web service execution (hosting environment), (2) the Web service usage (consumer/service integrator), (3) the Web service creation/modification (developer), and (4) business/domain related aspects of the Web service (provider).

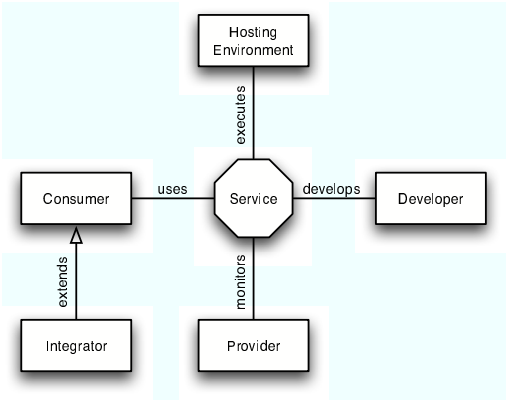

Figure 1. Factors that influence Web services.

The **hosting environment** is responsible for the execution of the Web service. It constitutes the software environment (e.g., operating system and Web service container) and the corresponding hardware on which the software is executed.

The **provider** of a Web service is responsible for providing domain or business expertise (e.g., service pricing, etc.) and specifying functional and non-functional requirements for Web services. Changes in business aspects and requirements have strong influence on the evolution of Web services.

The **developer** implements the Web service and makes the Web service actually run. The developer transforms the high level business perspective to a technical level. The developer writes technical specifications (interface descriptions) and the actual code of the Web service.

The Web service **consumer/integrator** uses a Web service to fulfill a certain task. Rather than paying attention to technical details, such as security and communication protocols, Web service consumers focus on QoS aspects, such as response time, availability, and reliability. The Web service integrator is similar to the consumer, but focuses on technical aspects as well. Interface descriptions, pre- and post-conditions of the Web service execution, are of concern for Web service integrators.

3.2 Information Sources

As mentioned in Section 2, current approaches to describe Web services cover different aspects of Web services. In our approach, we integrate Web service related information from various information sources into a common Web service information model (see Figure 2). This provides

us with a holistic view on Web services and establishes the foundation for our analysis of evolutionary changes of Web services.

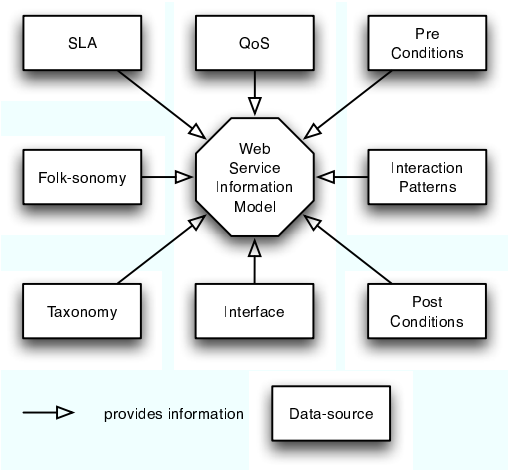

Figure 2. Data sources of the Web service information model.

The properties shown in Figure 2 are subject to change over the life-cycle of Web services. For instance, internal optimization such as refactoring the source code of a Web service and the use of new hardware can lead to changes in the response time of a Web service. We analyze the categories of Web service changes and the relations between changes in the next section.

3.3 Evolutionary changes in Web services

Every data change during the life cycle of a Web service is considered as evolutionary change. In our approach, we provide an extensible categorization for the classification of the changes a Web service can have over its lifetime (see Table 2). The classification of changes is the base for the analysis of Web service evolution. The combination of Web service change classification and the time of changes allows us to uncover the correlation of Web services changes. These correlations can be used to predict potential Web service behavior when certain changes happen. For instance, the usage of a Web service may decrease, if response time of the Web service increases (see Section 6 for more details).

3.4 Representation of Web Service Information Model

From our abstract model of existing information sources associated with Web services in Figure 2, we have developed a data representation model that is able to describe

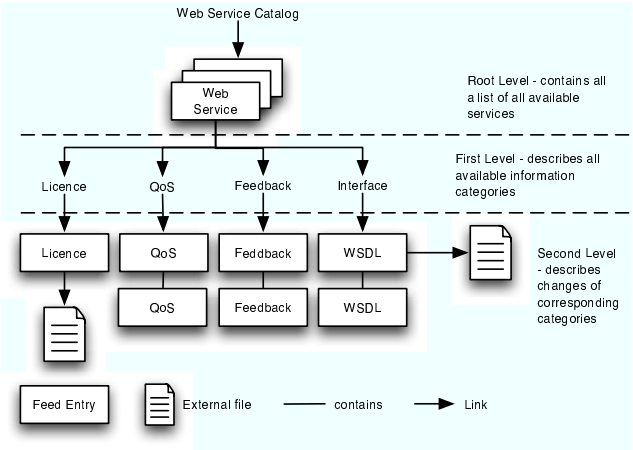

Figure 3. Representation of Web Service information model

those existing information. Figure 3 shows how Web service related information is organized in our data representation. We represent Web services information is in a hierarchy, starting with a `Web Service Catalog` which lists all available services.

Within a `Web Service Catalog`, concrete information can be stored internally in the body of the respective elements or linked to external source using URI. Every `Service Information` element has meta data specifying information category, given in Table 2. Therefore, based on this meta-data, tools know how to process the content of information associated with a `Service Information`, as well as how to obtain historical information within a category. URIs enable reuse existing tools and frameworks, and remain agnostic concerning the actual data model. More importantly, this mechanism allows us to distribute information of Web services into separate places as well as to easily integrate with external tools which provide Web services related information. In addition, we can extend the available Web service information model by adding new categories to our model.

The examples in Listing 4 and in Listing 5 (see Appendix) show how this meta information about the `WISIRISFuzzySearch` service is embedded in the corresponding (sub-)elements. We model meta information in `category` elements with links to external information, such as schemas, directories, etc. Listing 4 illustrates how general information about the Web service is encapsulated in two `category` elements, namely in (i) directory related information and (ii) versioning information. We refer to the current version of the interface using the `link` element.

We provide time-stamps to order the changes in the temporal space. Our approach is flexible enough to integrate arbitrary information (e.g. versioning, etc.) and to add cor-

Change category	Description	Trigger
Interface	Operations added/removed changes in operation signatures	Developer
Pre-conditions	Change of pre-conditions because of new interface or SLA	Developer, Provider
Post-conditions	Change of post-conditions because of new interface or SLA	Developer, Provider
Message exchange patterns	Change of protocol because of changes in the interface	Developer
Advertised QoS	QoS properties were modified	Provider
Measured QoS	Monitored QoS properties have changed	Consumer, Integrator
Hosting environment	Changes in the hardware environment of a Web service and in the Web service software execution environment	Developer
Implementation	Refactoring of source code of Web service	Developer
Consumer feedback	Consumer provided feedback for a Web service	Consumer, Integrator
SLA	Modification of existing SLAs or addition of new SLAs	Provider, Consumer
Documentation	The description of the Web service was changed	Provider, Developer
License	The license description of the Web service was changed	Provider, Consumer

Table 2. Evolutionary changes of a Web service during the life cycle of a Web service

responding meta information. This allows us to correlate changes of Web service interface descriptions with other information, for instance QoS. The example (Listing 6) in the Appendix shows two consecutive entries for QoS related information of a Web service. These entries show differences in the execution time of a Web service. Having this kind of information, we can search for instance for events that happened between these two observations to find an explanation for the behavior of the Web service.

3.5 Searching in SEMF

To search in the distributed database, we provide a XQuery interface. XQuery expressions allow for complex search criteria. For instance, it is possible to list all available Web services of a SEMF instances with its descriptions (see Listing 1).

```
declare namespace a = "http://www.w3.org/2005/Atom";
let $pattern := util:unescape-uri
(request:get-parameter("pattern","'1'='1'"), "UTF-8")
let $str := concat("for $x in
doc('ServiceCatalog/.feed.atom')
/a:feed/a:entry[",$pattern,"] return $x")
let $ws := util:eval($str)
return
for $x in $ws
return
<ServiceList>
  <Name> {data($x/a:title)}</Name>
  <Description>{data($x/a:category/@label)}<Description>
</ServiceList>
```

Listing 1. XQuery expression that returns a list of all available services

This is a foundation for analysis of the behavior of services during certain time intervals with regard to arbitrary criteria (see section 6 for a more detailed example). Our current prototype provides a simple Web based interface that allows to search for Web services using XQuery expressions.

4 Architecture of Service Evolution Management Framework

The management of evolutionary changes needs mechanisms to collect Web services information from various sources, managing and providing the information to different clients. Figure 4 depicts our service evolution management framework (SEMF) which supports our proposed Web service model from Section 3. SEMF is a distributed framework where every SEMF instance is responsible to collect the information locally and stores it in its XML database. However, one SEMF instance can store its data into another instance to distribute the content.

The *XQuery Interface* provides the means to execute arbitrary queries against the database. SEMF uses the XQuery interface that is provided by most existing XML databases. The *Plugin Manager* manages the plugins (see Section 4.1 for details). The *Atom Feed Generator* provides an Atom based RSS feed. Users can register for arbitrary feeds in order to get notifications of Web service changes. The *Syndication Module* allows the syndication of distributed Atom feeds into a single coherent Atom feed. The *Data Access Module* reads and writes Web services information from/to a local XML database.

4.1 SEMF Plugins

The information integration from different sources is based on extensible plugin mechanism. Given a type of information, many plugins can be developed to monitor Web services or to gather the type of information from other tools/frameworks. By employing different mechanisms, these plugins are responsible for collecting Web services related information, for example, using polling strategies (for instance, for the QoS plugin) or user input from Web browsers (e.g., user feedback and taxonomy information), and stor-

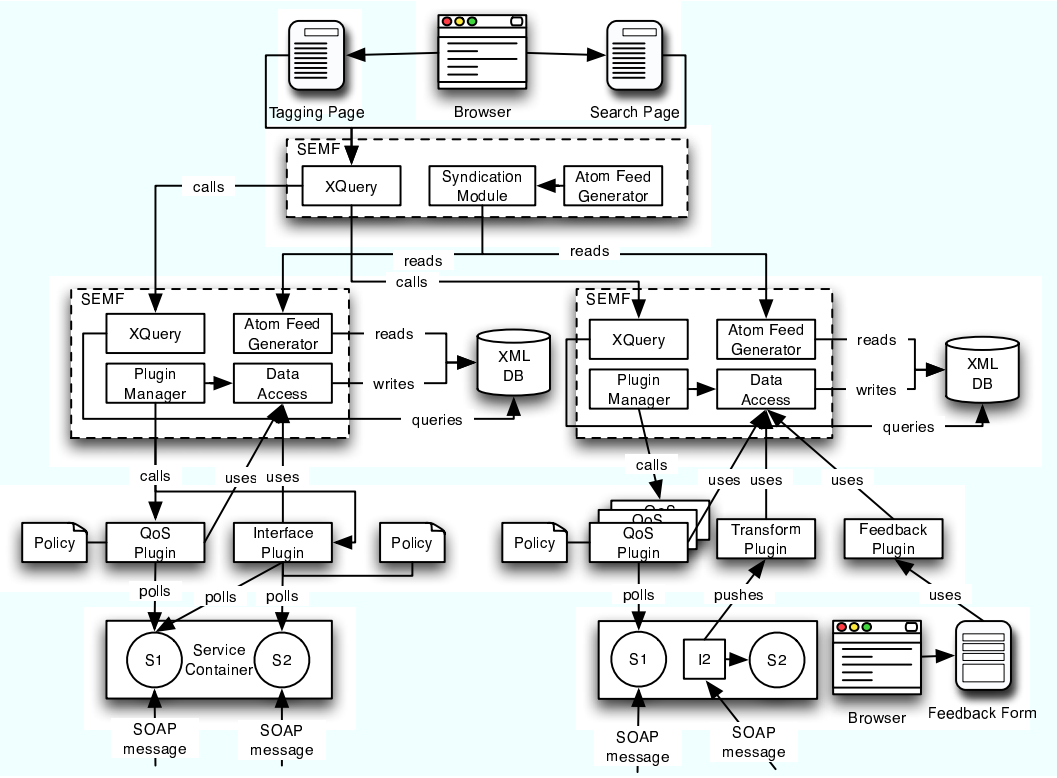

Figure 4. Overview of SEMF architecture

ing the collected Web services information into SEMF. In SEMF, we provide a generic approach with the definition of an interface, that every plugin must implement. The interface describes the basic operations that are necessary to write the a plugin in SEMF(see Listing 2).

```
public String getName();
public String getDataURL();
public void pollData();
public void setUpdatePolicy(PluginPolicy policy);
public Schema getSchema();
```

Listing 2. SEMF Plugin Java Interface

The actual data collection is controlled by policies that describe how often a data source writes data into the data model. The data collection policy (see Listing 3) defines the data collection interval, update frequency, etc.

```
<UpdatePolicy>
 <Begin>03.01.2007</Begin>
 <End>17.01.2007</End>
 <Field>Response Time</Field>
 <UpdateFrequency>
  <Type>Daily</Type>
   <From>07:00:00</From>
   <Until>19:00:00</Until>
   <Recurrence type="min">10</Recurrence>
   <UpdateType>Incremental</UpdateType>
 </UpdateFrequency>
</UpdatePolicy>
```

Listing 3. Web service information model update policy

As shown in the architecture, we consider explicitly pulling and pushing strategies for plugins. The latter requires that a plugin is informed about asynchronous events, such as for instance incoming SOAP messages. This approach requires that the service hosting environment supports message interceptors that notify the plugin about the occurrence of such asynchronous events and is obviously more intrusive.

4.2 Query and Subscription of Web Services Information

Since the information is represented in XML, any client can search for relevant information associated with particular Web services by defining requests in XQuery. As the content of Web services information in a `Service Information` feed can be internally kept within the feed or be linked to an external source using URI, and different contents may be represented by different languages, we do not support distributed or recursive search at this time of writing. SEMF searches the `Web Service Catalog` and returns the result met the request. Based on that, the

client can access external information sources and perform further requests based on meta-data information.

A particular type of Web services information can also be subscribed through a simple registration mechanism. The client has to provide a Web service endpoint and basic information based on meta-data such as `category` and time.

5 Implementation

Our prototype is implemented in Java and provides a Web service and a REST (Representational State Transfer) based interface for the management of Web services. We use eXist [15] to persist the management information. We utilize the eXist Atom servlet that provides a REST based interface to write Web service related information into the XML database. We encapsulate the REST based interface in a lightweight Java API to provide the functionality to register/unregister Web services. The Java API also supports the management of plugins. Currently, we are able to attach a plugin to a Web service and we have developed a management component that is capable to invoke the plugin according to the update policy to collect data of a Web service, based on a polling policy. This allows us to be as non intrusive as possible. However, our approach is also capable of handling asynchronous update policies. To ensure a "smooth" operation, we require a Web service that operates in Web service container like Apache AXIS that allows to intercept incoming SOAP messages of Web services and to log them. In order to keep the performance penalty as low as possible, we also foresee the possibility that the plugin keeps a local file for the logging information. The plugin collects the data during the activity of the Web service and stores the data later in the database. The plugin also transforms the content before writing it into the SEMF database. This approach involves an overhead of several milliseconds (depending on the usage of the Web service). Usually, this overhead can be neglected compared to the executions times of Web services (see Section 6).

The current implementation only supports a simple syndication strategy. Basically, all information is gathered from the different SEMF instances as Atom feeds and the content is integrated into a single Atom feed. In addition, we provide a basic Web based interface to browse and filter the content using XQueries (see Figure 5).

6 Case Study

In this case study, we experienced with a set of Web services for business reporting that are hosted by Wisur [1]. Wisur provides business reports to customers and other financial information of companies and consumer related information. Wisur provides a number of Web services (see Table 3) that cover these services. The Web services access a relational database that consists of 200 tables with a maximum of six millions of entries per table. These tables contain all business related information of companies and consumers.

The services are distributed on two separate Web servers. Every server provides an Apache Tomcat container for Web services and runs either consumer Web services or company Web services. Every Web service logs its activities (invocation time, execution time, etc.) in plain text files. Using our framework, we observed QoS information (execution time, availability), usage patterns (how often was a service used during a day), and changes of the Web service interfaces. To minimize the overhead for the production system, we analyzed the log files of the Web services offline. We exemplify the usage of our framework with `Company Search Service` and the `Consumer Report Service`. We observed QoS information, usage frequency, the interface of the Web services as well as pre- and post-conditions. To minimize the performance overhead, we utilized a plugin that analyzed the log files of the Web service once per day. The size of the log files was considerably small (approximate one megabyte of raw data per day). In addition, the plugin extracted only the relevant data from the log file which lead to a few kilobytes of data per day. The data was transformed by the plugin and was stored in the xml database. Figure 6 presents the execution time of the `Company Search Service` Web service during working hours. Figures 7, 8, and 9 illustrate the observation results for service interface, pre-condition, and post-condition, respectively. The `Consumer Report Service` was observed with a different focus, namely service execution time (Figure 11) and service usage (Figure 10).

During the observation period, new features for the `Company Search Service` were desired by one customer. The change of Web service interface, from `Version 1` to `Version 2`, was detected on Sept 3rd, as shown in Figure 7. In addition to the interface change, new customers were permitted to access to the Web service in two batches. Those change of pre-conditions were on Sep 6th and Sep 9th (Figure 8). The content of the database was updated twice with consumer data that led to changes of the post-conditions on Sep 6th and Sep 9th (Figure 9), since the Web service offered more information about consumers. As shown in Figure 6, the interface, pre- and post-condition changes had no immediate effects on the response times of the Web service. In addition, we analyzed the average service execution time of the `Consumer Report Service` (Figure 11) in combination with service usage (Figure 10). These remained rather constant in a certain time interval during the observation period with on excep-

[1] Wisur http://webservice.wisur.at/services

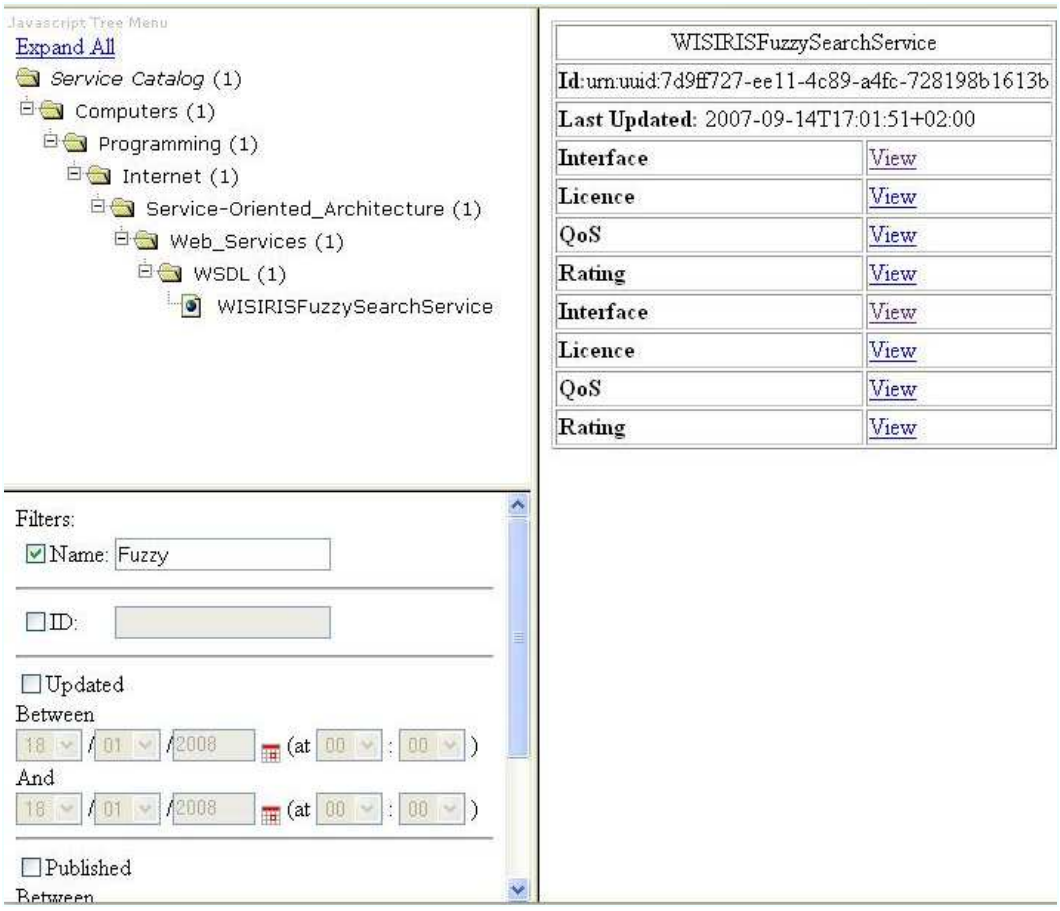

Figure 5. SEMF Web interface for browsing the Web service catalog.

tion. Both, service usage and execution time showed a single peak, but otherwise remained constant.

The observed data indicates that the back-end (i.e., the database) is already at its limits. When more data was added to the data base (Sep 6th and Sep 9th, see Figure 9 the execution times of the `Company Search Service` increased. This shows how Web services are affected by changes in the back-end, even if the interface remains stable. In addition, we see that an increased usage also leads to considerable changes in the Web service behavior as shown in Figure 6. When the customers started to use the new features beginning with Sep 12th, the response times of the Web service increased considerably. Generally speaking, more users mean more traffic and more data equals a greater execution time.

However, the single peak of the average execution time of the `Consumer Report Service` at October 1st not related to more service usage as show in Figure 10. In this case, there was an internal reorganization of the database. Or to be more specific: manual data cleansing by the employees, which had effects on the performance of the database and in turn on the average execution time of the service.

From the perspective of the customer, the only visible changes are obviously changes concerning the interface. However, while some changes remain transparent for the customer (database changes, more users), the observed behavior of the Web services (e.g., response time or post-conditions for higher hit rate when searching for companies or customers) changes. Such unexpected changes, especially when the service interface remains stable let customers wonder if the Web service is stable after all and whether the Web service can satisfy SLAs. Similarly, the integrator who builds a system using third-party Web services must rely on certain properties of a service so that the integration works in practice. When the response time suddenly doubles the service might not work properly in the integrator's system anymore, e.g., because of local time constraints. The provider desires to keep the system running at constant speed all the time in order to fulfill contracts with the customer. At the same time there is the desire to increase the usage of the services and to increase the revenue.

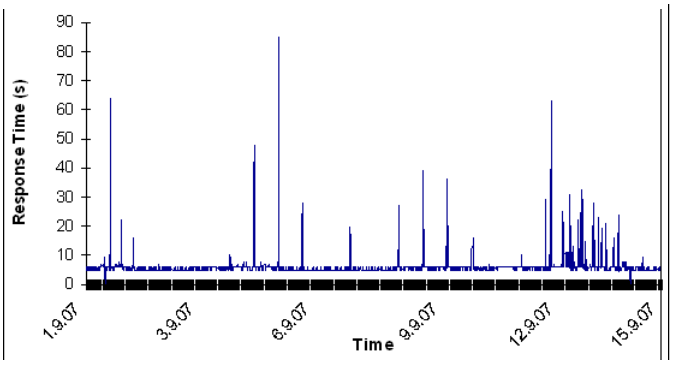

Figure 6. Measured response time

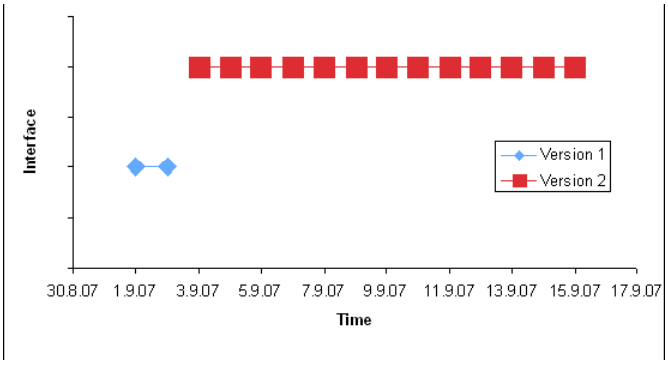

Figure 7. Changes in service interface

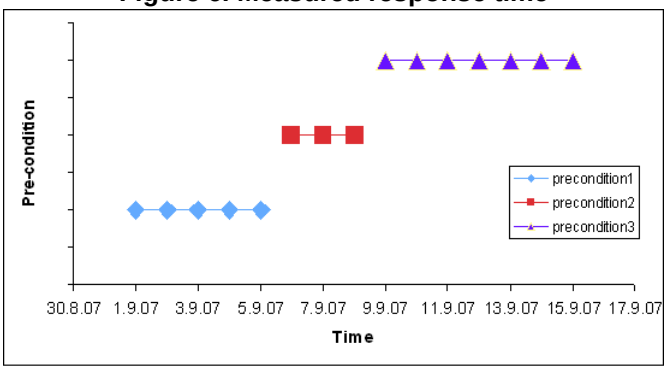

Figure 8. Changes in service pre-conditions

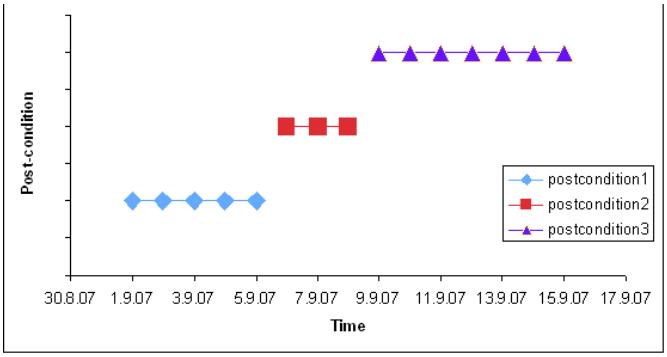

Figure 9. Changes in service post-conditions

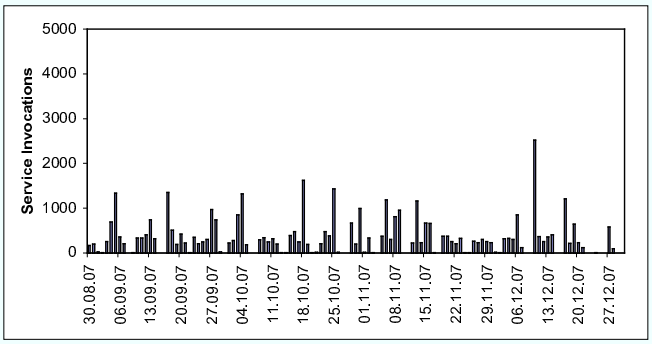

Figure 10. Service usage

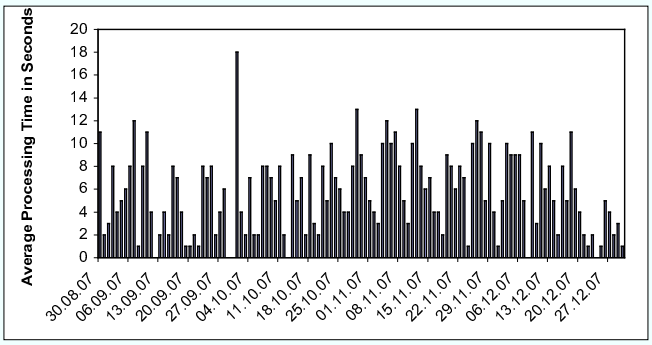

Figure 11. Average execution time

Service	Description
Company Search Service	lets customers search for companies in the Wisur database using different search criteria (e.g. name, address and register number).
Consumer Search Service	lets customers search for consumers in the Wisur database with fuzzy search criteria.
Consumer Report Service	generates consumer reports that include financial information.
Company Report Service	generates company reports that include financial information.
Company Creditworthiness Service	checks the credit worthiness of a given company and provides a rating in the range from 1(any credit possible) to 5 (no credit).
Consumer Creditworthiness Service	checks the credit worthiness of a given consumer and provides a rating in the range from 1(any credit possible) to 5 (no credit).
Consumer Address Service	provides information about historical addresses of a consumer.
Consumer Address Monitoring Service	notifies customer about changes of the consumers address.
Consumer Financial Monitoring Service	notifies customer about changes of the consumers financial situation.
Consumer Scoring Service	provides a statistical estimate (score) of a consumer that indicates the probability of financial problems within the next 12 month.
Company Scoring Service	provides a statistical estimate (score) of a company that indicates the probability of financial problems within the next 12 month.

Table 3. List of Web Services deployed at Wisur

With the data from the analysis, predictions about the behavior can be made. For instance, as shown by the data, new customers lead to longer service response times (see Figure 6). Moreover, the planing of maintenance activities can be adjusted to service usage in order to prevent unexpected execution times for customers.

The lesson we learned by this case study is that there is an urgent need to view Web services as dynamic entities. Static descriptions such as interfaces or ontologies are not sufficient when the Web services are used in production systems. Runtime aspects are important, and questions like how did the service behave in the past and what can be expected upon changes can be answered by looking at historical data.

Another aspect concerns potential applications that utilize our proposed framework. As outlined in Figure 4 we envision many different types of applications. For instance, a Web portal could use the framework in order to provide the facilities for user feedback, searching, tagging, etc. These applications are not limited to top level applications. We also consider the creation of plugins that provide the means for developers to attach information to Web services, such as comments, technical descriptions, diagrams, etc.

7 Conclusion and future work

In this paper, we introduced a Web service information model that integrates information of various data sources and presented SEMF, a framework that provides management features for the management of evolutionary changes of Web services. Our prototype architecture provides the necessary mechanisms to access and integrate the available information from distributed sources.

SEMF provides the foundation for the management/-monitoring of the evolution of Web services, based on that we will focus on the analysis of the evolution of Web services in a bigger context, and analyze in greater detail the dependency among changes of Web services. Moreover, we will concentrate on the full implementation of SEMF, its performance analysis, and tooling on top of SEMF.

8 Acknowledgements

The work is funded by the FFG as part of the ITEA project OSIRIS. The authors would like to thank Daniel Gomes for his work on the graphical user interface.

References

[1] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M.-T. Schmidt, A. Sheth, and K. Verma. Web Services Semantics – WSDL-S, 2005.

[2] G. Canfora and M. D. Penta. Testing services and service-centric systems: Challenges and opportunities. *IT Professional*, 8(2):10–17, 2006.

[3] F. Casati, E. Shan, U. Dayal, and M.-C. Shan. Business-oriented management of web services. *Commun. ACM*, 46(10):55–60, 2003.

[4] N. Catania, P. Kumar, B. Murray, H. Pourhedari, W. Vambenepe, and K. Wurster. Web services management framework, version 2.0, July 2003.

[5] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) 2.0, 2007.

[6] L. Clement, A. Hately, , C. von Riegen, and T. Rogers. UDDI Version 3.0.2, 2004.

[7] R. Dumitru, J. de Bruijn, A. Mocan, H. Lausen, J. Domingue, C. Bussler, and D. Fensel. Www: Wsmo,

wsml, and wsmx in a nutshell. *The Semantic Web - ASWC 2006*, pages 516–522, 2006.

[8] R. Fang, L. Lam, L. Fong, D. Frank, C. Vignola, Y. Chen, and N. Du. A version-aware approach for web service directory. In *ICWS*, pages 406–413, 2007.

[9] D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, A. Polleres, and J. Domingue. Wsml a language for wsmo. *Enabling Semantic Web Services*, pages 83–99, 2007.

[10] G. R. Gangadharan, M. Weiss, V. D'Andrea, and R. Iannella. Service license composition and compatibility analysis. In B. J. Krämer, K.-J. Lin, and P. Narasimhan, editors, *ICSOC*, volume 4749 of *Lecture Notes in Computer Science*, pages 257–269. Springer, 2007.

[11] M. IBM and HP. Web services resource catalog (ws-rc), May 2007.

[12] IBM and Microsoft. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-wsil/ws-wsilspec.pdf.

[13] A. Keller and H. Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *J. Network Syst. Manage.*, 11(1), 2003.

[14] D. D. Lamanna, J. Skene, and W. Emmerich. Slang: A language for defining service level agreements. In *FTDCS '03: Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, page 100, Washington, DC, USA, 2003. IEEE Computer Society.

[15] W. Meier. exist, January 2008.

[16] OASIS. Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1, August 2006.

[17] N. Thio and S. Karunasekera. Automatic measurement of a qos metric for web service recommendation. *aswec*, 00:202–211, 2005.

[18] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma. Management applications of the web service offerings language (wsol). *Advanced Information Systems Engineering*, pages 1029–1029, 2003.

[19] V. Tosic, K. Patel, and B. Pagurek. Wsol - web service offerings language. In *CAiSE '02/ WES '02: Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, pages 57–67, London, UK, 2002. Springer-Verlag.

[20] W3C. OWL Web Ontology Language Overview, 2004. W3C Recommendation 10 February 2004.

9 Appendix

```
<feed xmlns="http://www.w3.org/2005/Atom">
    <id><![CDATA[urn:uuid:4668e52e-cbb8-4840-8699-b94cbb6ae901]]></id>
    <updated>2007-12-07T18:28:42+01:00</updated>
    <link href="#" rel="edit" type="application/atom+xml"/>
    <title>Interface</title>
    <entry>
        <id>urn:uuid:8576e52e-cbb8-4840-8699-b94cbb6ae901</id>
        <updated>2007-09-14T16:37:53+02:00</updated>
        <published>2007-09-14T16:37:53+02:00</published>
        <link href="?id=urn:uuid:8576e52e-cbb8-4840-8699-b94cbb6ae901"
        rel="edit" type="application/atom+xml"/>
        <title>Interface</title>
        <link href="http://webservice.wisur.at:8000/axis/
        services/WISIRISFuzzySearchService?wsdl"
        rel="alternate" type="application/wsdl+xml"/>
        <category term="Interface"
        scheme="http://dmoz.org/Computers/Programming/
        Internet/Service-Oriented_Architecture/Web_Services/WSDL/"/>
        <summary>The service searches a relational database
        using fuzzy search criteria. The interface is exposed as
        WSDL definition. To use the Web service,
        a key must be downloaded from Wisur.</summary>
        <content type="application/wsdl+xml">
            <wsdl:definitions xmlns:impl="http://www.wisur.at/
            WISIRISFuzzySearchService/"
            xmlns:intf="http://www.wisur.at/WISIRISFuzzySearchService/"
            xmlns:apachesoap="http://xml.apache.org/xml-soap"
            xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
              targetNamespace="http://www.wisur.at/WISIRISFuzzySearchService/">
                    ...
            </wsdl:definitions>
        </content>
    </entry>
</feed>
```

Listing 4. WISIRISFuzzySearch interface.

```
<feed xmlns="http://www.w3.org/2005/Atom">
    <id>urn:uuid:3043abca-90a5-45d4-8508-bbaa4945ffad</id>
    <entry>
        <id>urn:uuid:121f3368-aea8-4197-86b0-f0561a428042</id>
        <updated>2007-09-14T16:37:53+02:00</updated>
        <published>2007-09-14T16:37:53+02:00</published>
        <title>Licence</title>
        <!-- link to current information -->
        <link href="http://wisur.at:8080/axis/
            /services/WISIRISFuzzySearchService?odrls"/>
        <category term="License"
            scheme="http://www.dmoz.org/Computers/
            Software/Licensing/"/>
        <category term="http://odrl.net/1.1/ODRL-EX-11.xsd"/>
        <content type="application+xml">
        <agreement>
            <context>
            <uid>urn:uuid:5321g52j-ffg8-6377-9001-g00cbb9ae111</uid>
            <date><fixed>2001-07-01T10:31:30</fixed></date>
            <pLocation>Vienna, Austria</pLocation>
        </context>
        <party>
            <context>
                <uid>urn:uuid:2334-g99j-ghg8-8711-9871-g74cbb9ae345</uid>
                <name>Wisur Gmbh</name>
                <reference>http://www.wisur.at</reference>
            </context>
        </party>
        <asset>
        <execute>
         <requirement>
          <peruse>
            <payment>
              <amount currency="EUR">1.00</amount>
              <taxpercent code="VAT">20.0</taxpercent>
            </payment>
         </peruse>
         </requirement>
        </execute>
        </asset>
        </agreement>
        </content>
    </entry>
</feed>
```

Listing 5. WISIRISFuzzySearch license information

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <id>urn:uuid:0d03c40b-104a-43e2-b85f-959c4cb5ce58</id>
  <title>QoS</title>
  <entry>
  <id>urn:uuid:a1612ce0-d8b6-11dc-95ff-0800200c9a66</id>
  <updated>2007-09-14T16:37:53+02:00</updated>
  <published>2007-09-14T16:37:53+02:00</published>
  <title>QoS</title>
  <link href="http://wisur.at:8080/axis/
  /services/WISIRISFuzzySearchService?qos"/>
  <category  term="QoS"
    scheme="http://www.dmoz.org/Computers/
    Software/Licensing/"/>
   <content type="xml">
    <QoS>
     <ExecutionTime>17</ExecutionTime>
     <Availability>100</Availability>
    </QoS>
  </content>
  </entry>
  <entry>
  <id>urn:uuid:127c2b28-086e-4992-8b44-b6b997889776</id>
  <updated>2007-09-15T16:37:53+02:00</updated>
  <published>2007-09-15T16:37:53+02:00</published>
  <title>QoS</title>
  <link href="http://wisur.at:8080/axis/
  /services/WISIRISFuzzySearchService?qos"/>
  <category  term="QoS"
      scheme="http://www.dmoz.org/Computers/
      Software/Licensing/"/>
   <content type="xml">
      <QoS>
       <ExecutionTime>19</ExecutionTime>
       <Availability>100</Availability >
      </QoS>
   </content>
   </entry>
 </feed>
```

Listing 6. WISIRISFuzzySearch Qos information