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 useful 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 Web services changes and (4) offering 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, resulting in a strong need to manage individual Web services and complex systems of Web services over the course of their lifetime. The management of Web services helps to answer questions such as (i) when and based on which information a Web service should be improved, (ii) which factors influence the employment of a Web service, and (iii) why a Web service is not widely used. Such questions are frequently asked by service developers, by the provider and also by 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 [14, 18, 15], and licensing information [9]. 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 [9]. However, there are diverse types of information that originate from various sources and different perspectives on Web services [2].

In particular, we address two main issues related to Web services: (i) what type of information is required for a given perspective, how to integrate all types of information into a single model, and (ii) how to manage those types of information. The first issue is related to the development of a novel aggregated, flexible and extensible information model for Web services. The latter is related to the development of a management framework. The framework deals with various types of information, whose management strategies are different (e.g., asynchronous or synchrnous updates), and keeps track of historical information. This paper contributes (i) a novel, common Web service information model that integrates heterogeneous information about Web services, and (ii) the design and implementation of SEMF (Service Evolution Management Framework), a distributed Web services management framework that addresses the aforementioned challenges.

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.

*The work is funded by the FFG as part of the ITEA project OSIRIS.

2 Related work

2.1 Web Services 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 languages used to describe Web services information

Existing Web service models cover different aspects of Web services such as service interface, QoS[17], SLA[14, 18, 15], and licensing information [9]. However, those models focus on only discovery issues. Table 1 overviews existing and adopted approaches and their descriptive capabilities. The main difference between our work and the aforementioned approaches (WSDL [5], OWL-S [20], WSMO [7], WSDL-S [1]) 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 Web service related information different sources.

In addition, what has not yet been provided by these proposed standards is the ability to tie together 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 [13] 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. The collection of run time information about Web services (usage statistics, logging, etc.) is discussed in [10]. Our framework integrates this kind of information and provides the means to analyze the available information.

Recently, WS-RC (Web Service Resource Catalog) schema [12] was introduced to describe management information of IT resources. In contrast to WS-RC 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. 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 to WSMF/WSDM, we concentrate on management information associated with Web services and their evolution.

The work in [8] discusses the management of service interface changes. 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 code optimizations of a Web service or new requirements to new operations). Based on our observations, we've identified four major factors of influence (see Figure 1). 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).

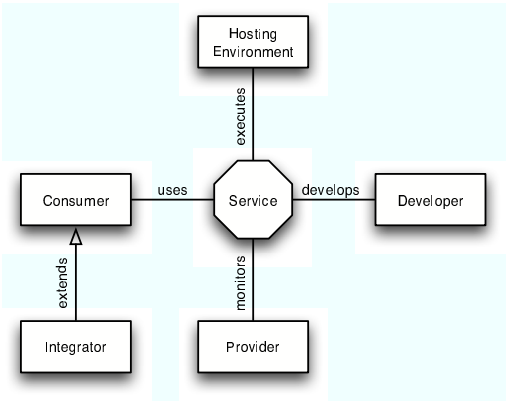

Figure 1. Factors that influence Web services

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

The **provider** of a Web service provides domain or business expertise (e.g., service pricing) and specifies 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

Web service changes caused by factors of influence are reflected by information changes. For instance, a new functional requirement leads to a new service interface. In our approach, we integrate these changes that originate 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.

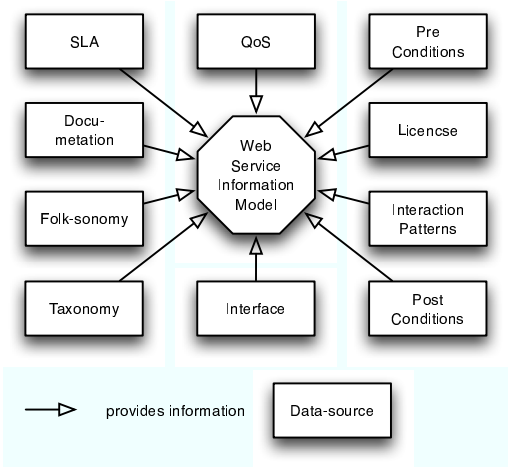

Figure 2. Data sources of the Web service information model

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 the response time of the Web service increases. Furthermore, the classifica-

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	Hosting environment
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 documentation of the Web service was changed	Provider, Developer
License	The license of the Web service was changed	Provider

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

tion makes it feasible to distinguish between information that is relevant for service developers, managers and users. Section 6 provides a detailed discussion about these issues.

3.4 Representation of Web Service Information Model

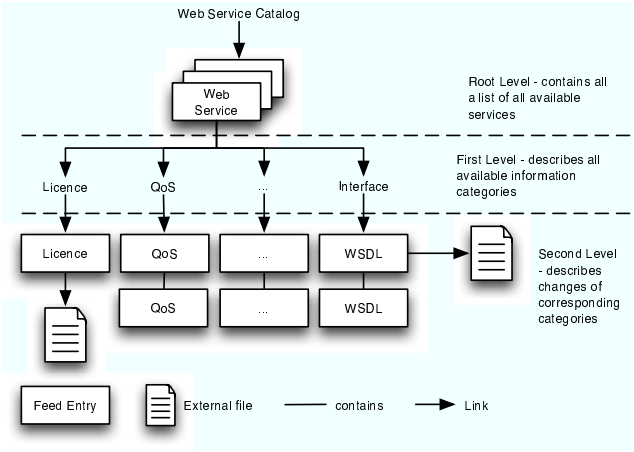

Figure 3. Representation of Web Service information model

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 aggregate those existing information. In order to have an extensible and flexible data model we utilized an Atom feed based representation of of the data. Figure 3 shows how Web service related information is organized in our data representation. We represent Web service information in a hierarchal Atom feeds, starting with a *Web Service Catalog* feed which lists all available services. As shown in the figure, we mapped every category that we introduced in table 2 (sub-)feed. Notice that, for the sake of brevity, we didn't include all available categories in the figure and present a simplified view of our data model. A Web service catalog includes a list of *Web services* feeds, each is used to store meta information that specifies different types of information given in Table 2. Every type of information is managed as a a list of entries ordered based on time stamps of the corresponding changes. An *entry element* describes the content of the Web service information (see Listing 1).

```
<id>urn:uuid:7d9f...</id>
<updated>2007-09-14T17:01:51+02:00</updated>
<title>WISIRISFuzzySearchService></title>
<entry>
 <id>urn:uuid:8576...</id>
  <updated>2007-09-14T16:37:53+02:00</updated>
  <published>2007-09-14T16:37:53+02:00</published>
  <title>Interface</title>
  <!-- link to current information -->
  <link href="http://www.wisur.at:8000/axis
           /services/WISIRISFuzzySearchService?wsdl"/>
  <category label="Directory information"
   term="Interface" scheme="http://dmoz.org/
      Computers/Programming/Internet/
     Service-Oriented_Architecture/Web_Services/WSDL/">
  </category>
  <category label="WSDL Versioning Scheme"
   term="Interface versioning"
   scheme="http://webservice.wisur.at/wsdl/versioning">
  </category>
   <content type="application+xml">
   <!-- wsdl description at 2007-09-14T16:37:53+02:00 -->
   <wsdl>...</wsdl>
  </content>
</entry>
```

Listing 1. WISIRISFuzzySearch interface.

As shown in the listing, we store the data either directly inside the entry element or refer to external data with links. Links to external information allow us to remain agnostic concerning the actual data model. In addition, the use of external links supports the distribution of information in different databases. Based on meta-data, tools know how to process the content of information associated with Web ser-

vices. We order the changes in the temporal space which allows us to correlate changes of Web service properties with each other. For instance, changes of interface descriptions may correlate with changes of QoS.

To search information in our model, XQuery language can be used [1]. For instance, it is possible to list all available Web services of a SEMF instances (see Listing 2).

```
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 $ws
return
<Name>{data($x/a:title)}</Name>
```

Listing 2. XQuery expression that returns a list of all available services

Note that XQuery in this case can be used not only by software applications but also advanced users or developers to access the information. However, this does not deal with the content of information described by semantics specification or via external link.

The proposed Web services information model 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).

4 Architecture of Service Evolution Management Framework

The management of evolutionary changes requires mechanisms to collect Web services information from various sources. 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 to store data in its XML database. However, one SEMF instance can store its data into another instance to distribute the content.

Depending on the role, there are different usages for our framework. For instance, a developer might use the XQuery interface to extract Web service related information concerning processing times, etc. A service provider can make use of a transformation a plugin that offers a web based representation of usage statistics or that generates spreadsheets for further statistical analysis of usage patterns, etc. A user of a Web service might use a feedback plugin that allows to provide feedback about the service. 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.

[1] http://www.vitalab.tuwien.ac.at:8000/exist/sandbox/sandbox.xql

4.1 SEMF Plugins

The information integration from different sources is based on extensible plugin mechanism. In SEMF, we provide a generic plugin 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 3).

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

Listing 3. 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 4) defines the data collection interval, update frequency, etc. Our approach supports pulling and pushing strategies for plugins to persist Web service relevant data in the database.

```
<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 4. Web service information model update policy

By employing different mechanisms, e.g. Java Server Pages and Apache Axis Handlers, SEMF plugins are capable of collecting Web services related information directly from users (e.g., user feedback, taxonomy information and SOAP requests) and to store this information in SEMF.

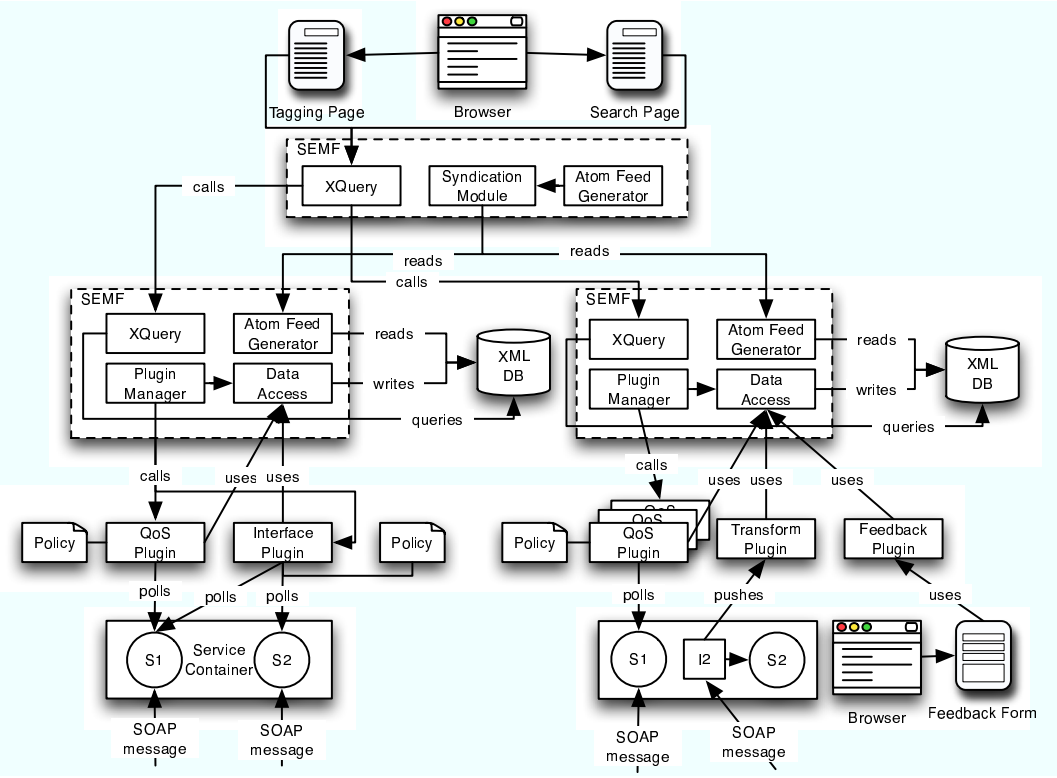

Figure 4. Overview of SEMF architecture

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. The content of Web service relation information in a `Service Information` feed can be internally kept within the feed. Alternatively, links to external sources using URIs, can are used. This way, we can support content represented by different languages. However, at this time of writing, we do not support distributed or recursive search. SEMF searches the `Web Service Catalog` and returns the result met the (XQuery-)request. Based on that, the client can access external information sources and perform further requests based on meta-data information.

5 Implementation

Our prototype is implemented in Java and provides a SOAP and a REST (Representational State Transfer) based interface for the management of Web services. We use eXist [2] to persist the management information. Our current implementation allows for the attachment of arbitrary plugins to a Web service. The management component is capable to invoke the plugin according to a polling policy to collect data of a Web service. This allows SEMF to be as non intrusive as possible. However, our approach is also capable of handling asynchronous update policies as well. In such cases, we require a Web service that operates in Web service container like Apache AXIS that allows to intercept incoming SOAP messages. In order to keep the performance penalty as low as possible, we require that the plugin collects the data during the activity of the Web service, keeps the collected data locally, and stores the data later in the database. This approach involves an overhead of several milliseconds per Web service invocation which can be neglected compared to the executions times of Web services (see Section 6). The syndication of distributed content is currently provided by a basic syndication scheme. In our current approach, all information is gathered from different SEMF instances as Atom feed 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 XQuery [3].

[2] http://exist.sourceforge.net/

[3] http://berlin.vitalab.tuwien.ac.at:8000/exist/wsc/catalog.xql

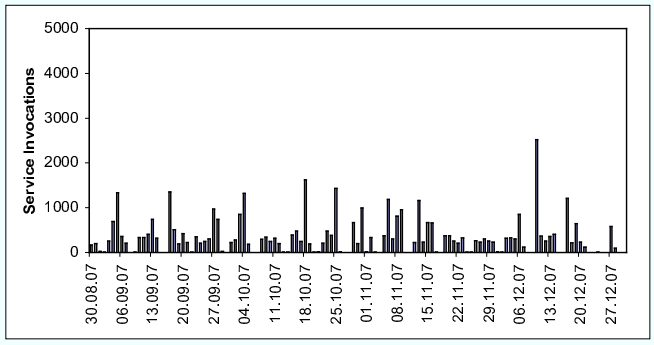

Figure 5. Service usage

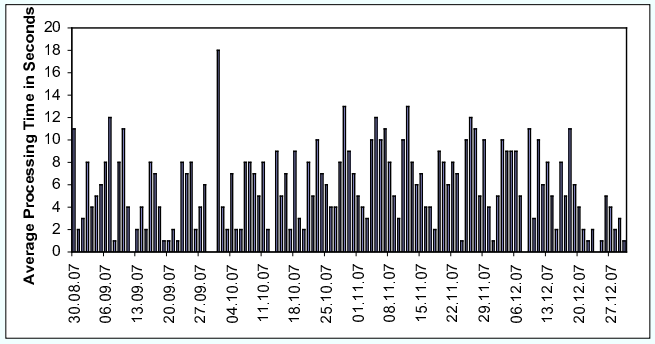

Figure 6. Average processing time

6 Case Study

In this case study, we experienced with a set of Web services for business reporting that are hosted by Wisur [4]. Wisur provides business reports to customers and other financial information of companies and consumer related information. 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 Web services that access the company and consumer data are distributed on two separate Web servers. Each 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 (processing time, etc.) in plain text files.

In the following, we illustrate the usage of our framework that was used to gather relevant information of Web services. We've deployed two instances of SEMF prototype, one at VitaLab [5] and one at Wisur. After a thorough investigation of publicly available Web services, we have registered 556 "real-world" services with or framework at VitaLab and monitor the interfaces of the registered services once per day. The monitoring process takes from 280 to 340 seconds to read the WSDL descriptions from 556 registered services and to store them in the database.

The second instance at Wisur uses dedicated plugins to extract relevant data from the log files of Wisur's Web services and to store the extracted data into SEMF. We observed QoS information (processing time, availability, etc.), usage patterns (how often was a service used during a day) and changes of the Web service interfaces of Wisur's Web services. We also provided a transformation plugin that converted XML data into comma separated files that were imported by open office to generate charts.

We now exemplify the use of our framework through a deeper analysis of Wisur's `Company Search Service`. In particular, we show how SEMF helps to identify changes of certain characteristics (e.g. interface) that affect other characteristics of the Web service (e.g. processing time). Figure 6 presents the processing time of the `Company Search Service` Web service during working hours during our observation period. The corresponding service usage is depicted in Figure 5. During the observation period, new features for the `Company Search Service` were desired by one customer. This led to an extension of the service interface with additional methods on September 3rd.

These new methods provide a different sort order of the search result as well as additional information of the companies that match the search criteria (a solvency indicator in addition to the company address). The addition of the new feature didn't have any impact on the processing time of `Company Search Service` as indicated by Figure 6. Considering the fact that the internal complexity of the service grew (more queries, more joins, different sort order) indicates that the hardware environment is capable of handling the higher service complexity well.

However, we observed a single peak of the service processing time on October, 1st. When comparing this with the usage data we observed no additional service usage. So the sudden change was not related to additional service usage. In this particular case, an internal reorganization of the database was the reason for the increase of the processing time.

In addition to the interface change, new customers were permitted to access to the Web service in two batches on September 6th and September 9th. Again, the extension of the user base didn't have any negative effect on the service in terms of processing time which supports the assumption that the hardware is able to handle a higher service load, before this has a measurable impact on the observable service processing time. The service invocation pattern (Figure 5) also showed no change in the overall usage patterns

[4] Wisur http://webservice.wisur.at/rss/WISIRISServices.rss
[5] http://berlin.vitalab.tuwien.ac.at:8000/exist/wsc/catalog.xql

of the users concerning the usage frequency that remained stable until 22nd of November where the use considerable declined. A comparison with historical data (not shown in the figure) indicated that this behavior is seasonal as well as the increase of the service usage towards the end of the year.

Another aspect that is illustrated by the use of our framework concerns potential applications that utilize SEMF. As outlined in Figure 4 we envision many different types of applications. Depending on the role there are different possibilities for applications. As already shown by the charts of our case study, statistical data can be put into graphs to provide an overview of the service behavior for service providers. Furthermore, we consider applications with plugins that allow developers to add technical information, e.g., (UML-)diagrams and how-tos, to services. And finally, we foresee the possibility of Web portals with facilities for user feedback, such as collaborative tagging[11].

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. In particular, we will investigate data mining techniques to assess changes of Web services and extend our Web service information model to incorporate this kind of information.

We are concentrating on the full implementation of SEMF, its performance analysis, and tooling on top of SEMF. We will focus on the investigation of the performance regarding multiple distributed plugins of SEMF. Furthermore, we are going to extend the graphical user interface and offer a plugin registry as well. In parallel, a powerful searching mechanism is being developed that will be able to search information described by different specifications in SEMF using XQuery. Furthermore, we intend to extend SEMF's service management capabilities beyond monitoring functionality. In particular we are going to look at issues like enforcement of SLAs and other related issues.

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] 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.

[10] C. Ghezzi and S. Guinea. Run-time monitoring in service-oriented architectures. In *Test and Analysis of Web Services*, pages 237–264. Springer, 2007.

[11] S. Golder and B. A. Huberman. The structure of collaborative tagging systems, 2005.

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

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

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

[15] 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.

[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.