On Analyzing Evolutionary Changes of Web Services*

Martin Treiber, Hong-Linh Truong, Schahram Dustdar

VitaLab, Distributed Systems Group
Institute of Information Systems
Vienna University of Technology
{treiber, truong, dustdar}@infosys.tuwien.ac.at

Abstract. Web services evolve over their life time and change their behavior. In our work, we analyze Web service related changes and investigate interdependencies of Web service related changes. We classify changes of Web services for an analysis regarding causes and effects of such changes and utilize a dedicated Web service information model to capture the changes of Web services. We show how to put changes of Web services into an evolutionary context that allows us to develop a holistic perspective on Web services and their stakeholders in a ecosystem of Web services.

1 Introduction

Web services undergo changes during their life time. This behavior is of particular interest when putting Web services in the context of Web service ecosystems [1]. With regard to Web services, a Web service ecosystem is the environment in which a Web service operates. The environment consists of different stakeholders that have interest in Web services and influence or control the life cycle of a Web service. To understand these ecosystems, we need to understand Web services with regard to evolutionary aspects since they are the central entities in the ecosystems. In particular, we need to analyze the artifacts that have impact on Web services, Web service ecosystems respectively, and investigate the reasons for changes of Web services.

Currently there is little support for Web service evolution, even though the evolution of Web services accounts for major development costs. The evolution of Web services involves changes of requirements, changes of the implementation, changes of the Web service semantics, changes Web service usage and so on. These activities originate from different stakeholders, such as developers, providers, users and service integrators that interact in Web service ecosystems.

In this work, we focus on the complexity of evolutionary Web service modifications. It's important for Web service providers to understand the prerequisites and the consequences of modifications of Web services in the light of limited resources (time, manpower, money). Current practices to describe Web services do not take these dynamic aspects into account. Approaches such as (WSDL [2], OWL-S [3], WSMO [4], WSDL-S [5]) primarily focus on interface related issues and do not model changes. We

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

argue that an evolutionary model requires a holistic perspective on Web services and needs to integrate information from various perspectives as well as different (design time and run time) data sources [6].

Real world Web services provide valuable input for our investigations with regard to the understanding of Web service evolution. In this paper we are using real world Web services of an Austrian SME called Wisur [1] to illustrate the challenges concerning Web service evolution. Wisur provides business related information to customers. Their main business is to sell business reports with information about companies (turnover, financial situation, etc) and consumer data (address, date of birth, etc.) to their customers. Wisur's customers use Wisur's Web services to search in Wisur's database and to access the desired information [2]. As soon as the requirements of customers change, Wisur needs to adapt their services accordingly. Examples are for instance the addition of new functions to existing services, or the creation of new Web services. In addition, Wisur reviews it's Web services once a month to look for issues that might cause problems in the future. For instance, a planned augmentation of a database with consumer data can lead to longer execution times of queries which have impact on the overall execution time of the Web service that queries data from this database. When putting these activity into the context of Web service evolution, we can observe the need for classification of modification activities under the umbrella of Web service evolution.In this paper, we focus on the evolution of single Web services that we consider as atomic building blocks of Web service ecosystems. In particular we analyze changes of Web services based on empirical observations. We investigate the impact of these changes from different perspectives. We propose a methodology that identifies influencing factors of Web services and a model of Web service related changes. Our major contribution is to define the basic element of Web service evolution, i.e. the *evolutionary step*, and embed it in Web service ecosystems. The rest of the paper is organized as follows. After summarizing related work in Section 2 we provide our analysis of changes of Web services in Section 3. We conclude the paper with a discussion and an outlook in Section 4.

2 Related Work

From an organizational point of view, our work is related to service management in general. Approaches such as WSDM [7] or WSMF [8] offer frameworks to collect data about Web services and to manage them. The work of Casati et. al. [9] focuses on the business perspective of Web service management. Fang et. al. [10] discuss the management of service interface changes. The authors describe version-aware service descriptions and directory models. The work by Kaminski [11] introduces a framework that supports service interface adaptors for backward compatible versioning of Web services. Our work covers different aspects of service changes, such as runtime aspects (e.g., QoS), implementation changes, semantic changes, etc.

The collection of run time information about Web services (usage statistics, logging, etc.) is discussed by Ghezzi et. al. [12]. The area of Software Configuration Manage-

[1] http://www.wisur.at

[2] see http://webservice.wisur.at:8000/axis/services/WISIRISFuzzySearchService?wsdl for an example of a Web service

ment [13] is also related to our research with regard to analysis of changes of software systems and versioning issues. We follow ideas such as the collection, classification and monitoring of change events from the aforementioned approaches. However, our approach differs insofar, since we focus on evolutionary aspects that are not covered by management approaches.

From an evolutionary perspective, our work is closely related to evolutionary aspects of software in general. Of particular importance is the work of in [14] where the authors studied software changes and considered software systems as dynamic entities that change over time. Lessons can also be learned from the basic SPE classification scheme which is extensively discussed in [15] and [16].

Papazoglou [17] discusses the challenges of service evolution and introduces two categories of service related changes. While similar in spirit, our work follows a broader scope, since our approach introduces a concrete information model to capture service related changes and lays the foundation for deeper analysis of service related changes. Andrikopoulos et. al. [18] discuss the management of service specification evolution. The author's abstract service definition model addresses Web service evolution on an abstract layer whereas we follow a bottom up approach with the analysis of single Web services with regard to Web service evolution.

The work in [19] describes the use of a dependency structure matrix that reflects the overall software structure and highlights dependency patterns between software components. Our work includes the aspect of dependency, however, we focus on a broader set of information in our analysis (e.g., QoS information, usage data).

3 The Anatomy of Web Service Changes

In this section, we analyze changes that are related to Web services. As indicated by the working example of the introduction, changes of Web service happen due to various reasons. During our observations, we've developed a methodology that classifies the factors of influence associated with Web services (see Table 1). We base our classification of the factors of influence on the work of Canfora and Penta [20] that identified the different stakeholders of Web services. Our proposed methodology consists of the following steps:

1. Identify the stakeholders that are interested in the Web service. This is done by analyzing the development process of Web services and investigating how the communication process is structured. For instance, developers might be informed about changes of Web service requirements by phone or by email.
2. Identify the tasks of the corresponding stakeholders. This step is basically the assignment of responsibilities to the interested parties. For instance, a user of a Web service is responsible for giving feedback about the service performance.
3. Collect data of Web services and identify the source of the data. In this step, we distinguish between runtime and static data. Runtime information (e.g., QoS, Web service usage statistics) is provided by tools that continuously monitor Web services. Other Web service related information is (e.g., requirements, user feedback) is entered into a persistence framework by developers, etc.

Perspective	Task
Provider	The provider is responsible for the planing and conception of the Web service. This includes the definition of requirements, the negotiation of SLA with customers, service pricing policy, etc. as well as managing changes of these.
Developer	The main task of the developer is the implementation of the Web service. The developer needs to manage changes of interface descriptions, different versions of the implementation and track change of requirements.
Service Integrator	The service integrator's task is to integrate external services into a software system and focuses on technical aspects of services. Service integrators are interested in changes of the interface, QoS attributes and the semantics of the Web service since these have effects on the integration of the Web service. Service integrators also modify the requirements of Web services.
User	The end user of a Web service is the actual consumer of a Web service and has interest in the functionality of Web services from a non technical perspective. The end user specifies the functional requirements and defines QoS attributes that must be satisfied by a Web service.
Broker	The Web service broker manages information of different Web services in a repository that is publicly available for searching and browsing.

Table 1. Perspectives on Web services

In our previous work [6], we've analyzed information sources concerning changes of Web services and introduced a hierarchical information model to store this kind of data as Atom feeds (see Figure 1)[3].

Based on these empirical observations, we've identified dependencies between different perspectives and generated a common generic model for a general classification of the available information (see Figure 2).

During the evolution process of a Web service, we can observe transitive effects of changes that lead to a new versions of a Web service (see Figure 3). We consider this change propagation as foundation for the understanding of evolutionary changes of Web services. In this respect, we regard a set of interrelated modifications as step in the evolutionary process of Web services.

To illustrate this, let us consider the following illustrating scenario: The provider of a commercial Web service monitors its service continuously in order to obtain usage statistics of Web service. Critical changes in the usage pattern, like a drop in the daily use of a commercial service are very important for the provider. In such cases these changes trigger activities of the Web service provider. For instance, the provider might contact the customer and ask for feedback. Let's assume that the customer is not satisfied with the pricing of the service with regard to its performance which is the reason for a change in the service usage. The customer might argue that competitors are able to provide a similar service with a lower price. Since the provider does not want to lose the customer, the provider adapts the pricing and updates the corresponding SLA. Meanwhile the developer was informed by the provider that a customer is not satisfied and that the SLA have changed. The provider requests to optimize the service performance

[3] see http://berlin.vitalab.tuwien.ac.at/projects/semf for detailed information about the Web service information model.

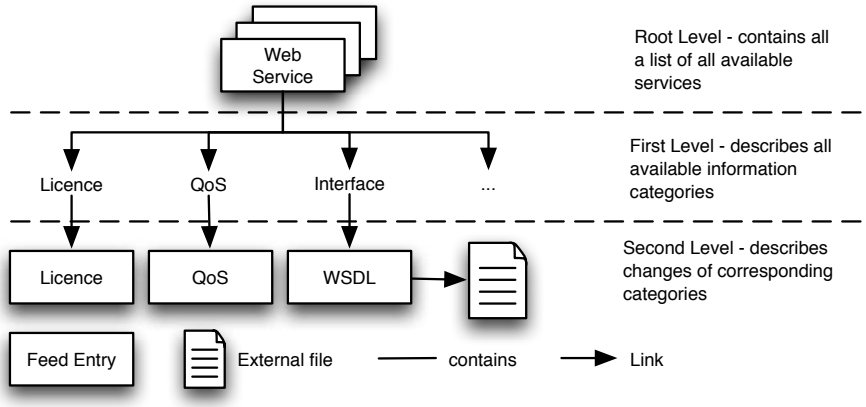

Fig. 1. Web service information model.

because the provider expects in the future customers to require better performance of the service. These changes have effect on the QoS parameters of the Web service which in turn influence the usage of the Web service. As shown by the example above, we can observe change propagations and impacts on different perspectives from one single change event. We've summarized potential changes and impacts in Table 2.

In the following subsections we discuss major changes of Web services in detail. We show how change activities are interrelated and explain how these activities contribute to the evolution of Web services. We provide examples that are represented in our service evolution and discuss the benefits of out approach for the different stakeholders of Web services. Notice that we group changes into two groups. We consider dynamic changes that occur during runtime. These changes can be observed by monitoring tools that log performance related QoS attributes (e.g., response time, throughput, etc.) and collect data of the use of a Web service. Static changes happen prior to the execution of a Web service and may be triggered by observations of the run time behavior of a Web service. The trigger for static changes may be either changes in the observed behavior (decreasing response time due to more server load) or changes in the requirements such as requests for new functionality, etc.

3.1 Web Service Requirements Changes

Changes of requirements are the main driver for all evolutionary Web service changes. Requirements serve as "benchmark" for the correct functionality of Web service implementations. Changes of requirements are thus very critical during the evolution of Web services and have a number of effects on Web service characteristics:

Implementation Changes of requirements affect the implementation of Web services, since changes of Web service functionality need to be implemented by the developer.

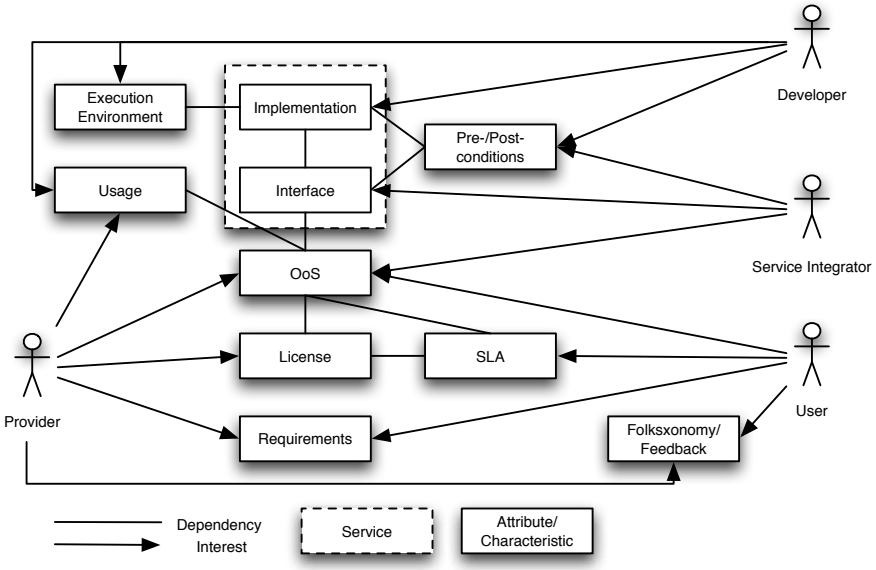

Fig. 2. Information model of Web service changes. Note that the dependencies between the different attributes are not mandatory and depend on the concrete service.

Interface The interface reflects changes of Web service requirements when these changes affect functionality of the Web service.

SLA The provider of the Web service may change SLAs with customers when their requirements change. For example, new functionality needs to be specified in SLAs as well as changes of the required performance of the Web service, data quality, costs, etc.

Pre- and Postconditions With changing requirements the prerequisites for the execution of Web services might change. For example, new requirements may require a registration for customers prior to the use of the Web service. The effects of the execution of Web service may also change with new requirements. For example, a data Web service might provide additional information to the customer.

All stakeholders of Web services have interest in Web service requirements. Conceptually, requirements can be considered as logical link that links different aspects of Web service modifications together. During the evolution of Web services, every evolutionary step is delimited by the definition of requirements and the publication of a new Web service. Activities by stakeholders as modification of the interface (developer), implementation (developer), definition of SLAs (provider, user), feedback (service integrator, user) are triggered by the definition of requirements, changes respectively.

Example. As noted before, a change of a requirement triggers a set of related activities in order to implement the requirement. Utilizing our evolution framework we can track

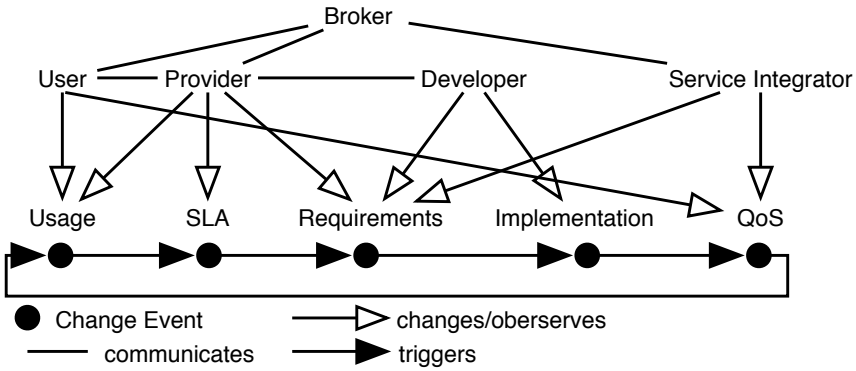

Fig. 3. Propagation of changes and interested parties

these activities and link them. When put into a historical perspective, provider can analyze, based on historical information, the costs of the different Web service versions when following the provided links to details about the implementation (see code snippet in Listing 1.1).

```
<change type="Requirement">
  <link>http://webservice.wisur.at/WISIRISFuzzySearchService?ReqPhon.pdf</link>
  <category>New Function</category>
  <description>A new function was added to the requirement</description>
  <cause type="Feedback">
    <reason>Customers want to search with phonetic methods.</reason>
    <trigger type="User">...</trigger>
  </cause>
<dependencylist>
  <dependency type="Implementation"><!— link to impl. change description —>
  <link id="urn:uuid:e2e2f679-8a67-439a-a65e-bbafd1dd0091"/>
  </dependency>
</dependencylist>
<impact><perspective type="Developer"/>...</impact>
</change>
```

Listing 1.1. WISIRISFuzzySearch requirement change

3.2 Interface Changes

Syntactical descriptions of Web services define available operations and the structure of messages that Web services are capable to process. We consider interface changes as (i) the addition of new functionality or (ii) the update of existing functionality (e.g., interface refinement with new parameters or removal of functionality). As shown in the overview table, the trigger such changes can be either the provider, the service integrator or the user of a Web service. The cause of an interface change is a change of the

Observed Change	Trigger	Impact on	Modification of	Effect on
Interface	Provider, User, Service Integrator	Integrator, Developer	Implementation	QoS, SLA, Usage
Implementation	Developer	Integrator, User	Implementation	QoS, Interface
QoS	Usage	Provider, Developer	Implementation, Interface	Interface, QoS, SLA, Usage
Usage	User	Provider	Contact user	SLA, QoS
Requirement	User, Integrator	Provider, Developer	Interface, Impl., SLA	Usage
SLA	Provider	User, Developer, Integrator	Usage	Requirement, QoS, Impl.
Pre-Post Conditions	Provider, Developer	User, Integrator	SLA	Impl.
Feedback	User, Integrator	Provider, Developer	SLA	Usage

Table 2. Impacts on Web service changes.

requirement. Consequently, interface changes affect primarily the service integrator that is using the service in other software systems, since s/he must adapt the software system that uses the service accordingly.We can observe the following effects of interface changes:

Implementation Interface changes have impact on the implementation of a Web service. Depending of on the type of interface change, we can observe different effects on the implementation of a Web service. The addition of new functionality or the update of existing functionality are reflected by modifications of the implementation.

QoS QoS attributes of Web service may be effected by interface changes. However, an interface change does not have immediate consequences for QoS properties. For instance, if the addition of new service functionality also changes existing implementation (e.g., optimizations) then the QoS attributes (e.g., response time, etc.) also change.

Pre- and Postconditions Changes of the interface have effects on pre- and post-conditions of Web services. These reflect the necessary conditions that must be fulfilled to execute a service. For instance, the update of existing functionality to a Web service might require new constraints to be satisfied, such as the provision of a customer identifier and a trial-key.

Usage Changes of the interface influence the usage of a Web service. Unless the interface change is backward compatible, a new interface with new functionality has impact on the usage of the service.

During the evolution of Web services, each publicly available interface version denotes a new version of the Web service. By analyzing the frequency of interface changes, Web service stakeholders are able to establish the interface stability of the Web service.

Example. From the perspective of the developer it is important to connect interface changes with requirements. In this way it is possible to check implementations for their consistency with (functional) requirements. By combining versioning information with interface modifications, developers are able to track different service versions and corresponding requirements.

From the perspective of the service integrator interface changes are very critical since the service integrator relies on stable interfaces when integrating external services. When interfaces change, service integrators require Web services to be compatible with existing systems. If this is not the case, service integrators require information about the nature of the interface changes to infer how much they have to change. In our approach we follow the classification schema by Leitner et. al.[21] to classify interface related changes.

The example in Listing 1.2 illustrates how we integrate the changes into our service information model and how we link changes dependencies. In the code snippet, we show how we represent the addition of new functionality to a Web service.

```
<change type="Interface">
  <category type="Add Method"/>
  <description>A new search method was added to the
  WISIRS Fuzzy Search Service</description>
  <cause type="Requirement">
  <— link to the requirement where details can be found —>
   <link id="urn:uuid:823157f7-7174-4b09-b815-64750b0e2f83"/>
  </cause>
<dependencylist>
  <dependency type="Implementation">
<!— link to implementation information in SEMF —>
<link id="urn:uuid:912ae1a0-96b0-11dd-ad8b-0800200c9a66"/>
  </dependency>
</dependencylist>
<impact>
  <perspective type="System Integrator"/>
</impact>
</change>
```

Listing 1.2. WISIRISFuzzySearch interface change

3.3 Web Service Implementation Changes

Closely related to interface changes are implementation changes. We consider two types of changes, (i) code refactoring/internal optimizations and (ii) change in the functionality. The former subcategory are changes that are transparent for all users of the Web service. The latter is a consequence of interface changes since the new service functionality is expressed by interface changes. Potential triggers for implementation changes are changes of requirements which are caused by the service provider, the user or the service integrator.Implementation changes have effects on the following attributes of Web services:

Interface Depending on the type of implementation change, we can observe different effects of implementation changes on the service interface. The interface of a service changes when new functionality is added to the Web service or removed from the Web service. Code refactorings or internal code optimizations leave the interface of a Web service untouched since the functionality remains unmodified.

QoS Internal optimizations have effects on QoS. Consider for instance a database that is accessed by multiple parallel threads simultaneously instead of a sequential manner. This optimization changes the service execution time and is reflected by changes of the response time of the service. Similar, the addition of security mechanisms (e.g. WS-Security, etc.) to a service have impact on QoS attributes.

Pre- and Postconditions Both, pre- and postconditions are potentially affected by implementation changes. Depending on the type of implementation change, new service functionality (e.g., new methods to search in the provider database) obviously requires new pre conditions (e.g., new input parameters) that must be satisfied in order to execute a service. Postcondition changes depend on the type of implementation change. Consider for example a service that required payment and is now free of charge for customers. In this case, the service implementation was changed in order to acknowledge the new form of service use.

Usage The effects on the usage of Web services depend on the type of implementation change. Similar as in the case with interface changes, new functionality can lead to an increased usage of a Web service, because potentially more users can be served. Internal changes (along with enhanced performance) might also result in a higher usage of the Web service.

In our evolutionary approach, every evolutionary step is preceded be a series of implementation changes to achieve the fulfillment of requirements. The publication of a new Web service version denotes the finalization of all necessary implementation changes.

Example. As in the case of interface changes, the developer needs to trace modifications in the source code with respect to changes of requirements and user/service integrator feedback. In particular, when the developer modifies the implementation to improve the performance the developer needs to know whether the changes have the desired impact and requires feedback from the user/service integrator. From the perspective of the Web service provider it is important to know how much time was spent to implement in the required modifications. To illustrate these types of change, consider for example a Web service that offers facilities to search in a consumer database. In the example, internal changes were implemented that had no effect on the interface of the Web service. The code snippet in Listing 1.3 illustrates how we capture these information in our Web service information model.

3.4 Web Service QoS Changes

QoS related changes of Web service depend on changes of other properties of Web services. In contrast to implementation modifications, QoS changes are observed at run time. The reasons for QoS changes are manifold: server load, number of concurrent

```
<change type="Implementation">
<category type="Internal Modification"/>
 <description>The ordering of the search result was changed.</description>
  <cause type="Feedback">
    <reason>User require a ordered search result (by familyname).
   </reason>
    <trigger type="User">... </trigger>
  <link>webservice.wisur.at/</link>
 </cause>
<dependencylist> <— implementing class —>
 <dependency type="Class">
 <name>WISIRISSearchWrapper</name>
<description>Modification of SQL query</description>
  </dependency>
</dependencylist>
<version number="1"> <!— versioning information —>
  <effort developerid="12"> <!— implementation effort —>
  <hours>3</hours>
  </effort>
</version>
<impact><perspective type="Developer"/></impact>
</change>
```

Listing 1.3. WISIRISFuzzySearch implementation change

users, performance of back end systems such as databases, external factors such as network latency, network throughput, as well as issues like security, etc., influence the QoS attributes of Web services. Domain related QoS attributes, like data quality (completeness, correctness, etc.) when providing data centric services are also of concern. For instance, the hit rate of a search Web service is of importance when a provider desires to sell business reports. Simply put: the higher the hit rate, the higher is the probability that a user will use the service.We can observe the following effects QoS changes:

Usage Changes of Web service related QoS have impact on the usage of Web services. When a service is selected by QoS attributes like response time, then a degradation of QoS changes such as a higher response time can lead to a reduced service usage.

Implementation Observed QoS changes may lead to implementation changes. Internal optimizations of the program code (e.g., different algorithms) are potentially used to enhance performance related QoS attributes.

During the evolution of Web services, QoS attributes serve as indicator concerning the overall *fitness* of the Web service. With QoS information, we are to measure the fitness of Web services with regard to SLAs. When put into a historical context, QoS data provides information about the overall development of a Web service and allows to estimate when the performance of a Web service may become critical.

Example. With regard to the provision of data centric services, we address (i) data quality (is the provided information up to date? and (ii) typical QoS (response time, availability, etc.) of a Web service. We now show an example that highlights service quality aspects from the perspective of the service provider with regard to service performance. The code snippet in Listing 1.4 shows a notification about the violation of SLA constraints that is generated by a monitoring tool that logs the performance of

Wisur's Web services, making the observation of QoS very important from the perspective of the service provider.

Notice that our tool includes information for the developer in order to track the part of the Web service implementation which is responsible for the violation of the SLA.

```
<change type="QoS">
  <category type="Violation"/>
  <cause type="Usage">
    <reason>Response of WISIRISFuzzySearchService</reason>
    <trigger type="Service Environment">...</trigger>
  </cause>
<dependencylist>
<dependency type="SLA"><!— link to sla information in SEMF —>
  <link id="urn:uuid:da66f3c0-96da-11dd-ad8b-0800200c9a66"/>
 </dependency>
</dependencylist>
<impact>
  <perspective type="Developer"/><perspective type="Provider"/>
</impact>
<details>
  <classes><class name="WISIRISDataAccess">
  <executiontime>59955ms</executiontime>
  <exception/>
  </class></classes>
</details>
</change>
```

Listing 1.4. WISIRISFuzzySearch QoS change notification

Similar to Web service providers, end users are concerned with the Web service quality. Consider the example, of a Web service which must respond within 60 seconds and be available 24/7. The data presented in Figure 4 and Figure 5 shows the observed execution times of the reporting service of two consecutive months of a real world Web service from Wisur. As shown in the figure, the execution time during April 2008 was constantly under 30 seconds, with a tendency to increase towards the end of the month and a constraint violation at the end of April 2008. This lead to user feedback and triggered a change in the implementation of the Web service. The observed execution time in May 2008 was considerably higher (more peaks moving towards 60 seconds) but no constraint violation occurred.

4 Discussion and Outlook

In this paper, we analyzed dependencies of Web service changes and provided a model that captures the changes. We introduced a conceptual model which offers the means for deeper analysis of these changes. In context of Web service evolution we are able to define an *evolutionary step* as set of activities (modifications of the interface, implementation, requirements, SLA) that are triggered by different stakeholders of Web services. The result of an evolutionary step is a new version of a Web service that is adapted to these changes.

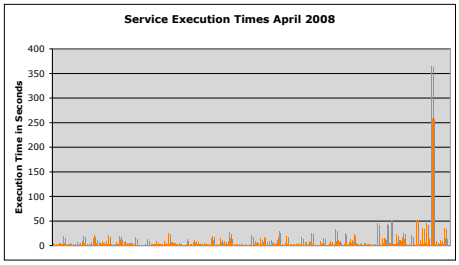

Fig. 4. Observed execution during April 2008

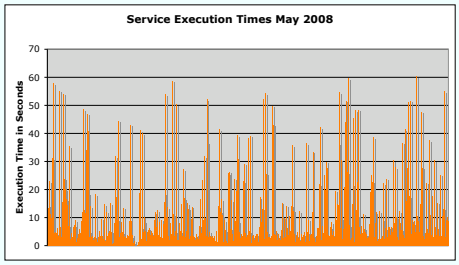

Fig. 5. Observed execution during May 2008

This lays the foundation for the Web service evolution process. We consider Web service evolution as an (potentially) *indefinite sequence of evolutionary* steps that result in observable changes of the Web service. We assume that there are several *variations* of a Web service at a given point in time. Every variation represents a independent evolution sequence of a Web service and is represented by historical information.

In future work, we will focus on composite Web services and investigate evolutionary issues of Web service compositions and investigate graphical models for the representation of the evolution [22] of complex composite Web services. Moreover, we are going to formalize our proposed conceptual methodology with a meta model that provides a formal foundation for roles and change dependencies. Furthermore, we will investigate complex event processing with regard to evolutionary aspects. In this context, we plan to extend our framework with the support of event processing in the context of Web service registries as discussed in [23].

References

1. Barros, A.P., Dumas, M.: The rise of web service ecosystems. IT Professional **8** (2006) 31–37
2. Chinnici, R., Moreau, J.J., Ryman, A., Weerawarana, S.: Web Services Description Language (WSDL) 2.0 (2007)
3. W3C: OWL Web Ontology Language Overview (2004) W3C Recommendation 10 February 2004.
4. Dumitru, R., de Bruijn, J., Mocan, A., Lausen, H., Domingue, J., Bussler, C., Fensel, D.: Www: Wsmo, wsml, and wsmx in a nutshell. The Semantic Web - ASWC 2006 (2006) 516–522
5. Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.T., Sheth, A., Verma, K.: Web Services Semantics – WSDL-S (2005)
6. Treiber, M., Truong, H.L., Dustdar, S.: Semf - service evolution management framework. In: SEAA 2008. (2008) to appear
7. OASIS: Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1 (2006)
8. Catania, N., Kumar, P., Murray, B., Pourhedari, H., Vambenepe, W., Wurster, K.: Web services management framework, version 2.0 (2003)
9. Casati, F., Shan, E., Dayal, U., Shan, M.C.: Business-oriented management of web services. Commun. ACM **46** (2003) 55–60

10. Fang, R., Lam, L., Fong, L., Frank, D., Vignola, C., Chen, Y., Du, N.: A version-aware approach for web service directory. In: ICWS. (2007) 406–413
11. Kaminski, P., Müller, H., Litoiu, M.: A design for adaptive web service evolution. In: SEAMS '06: Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, New York, NY, USA, ACM (2006) 86–92
12. Ghezzi, C., Guinea, S.: Run-time monitoring in service-oriented architectures. In: Test and Analysis of Web Services. Springer (2007) 237–264
13. Conradi, R., Westfechtel, B.: Version models for software configuration management. ACM Comput. Surv. **30** (1998) 232–282
14. Lehman, M.M., Ramil, J.F.: Software evolution: background, theory, practice. Inf. Process. Lett. **88** (2003) 33–44
15. Cook, S., Harrison, R., Lehman, M.M., Wernick, P.: Evolution in software systems: foundations of the spe classification scheme: Research articles. J. Softw. Maint. Evol. **18** (2006) 1–35
16. Lehman, M.M.: Laws of software evolution revisited. In: EWSPT '96: Proceedings of the 5th European Workshop on Software Process Technology, London, UK, Springer-Verlag (1996) 108–124
17. Papazoglou, M.: The challenges of service evolution. Advanced Information Systems Engineering (2008) 1–15
18. Andrikopoulos, V., Benbernou, S., Papazoglou, M.: Managing the evolution of service specifications. Advanced Information Systems Engineering (2008) 359–374
19. Sangal, N., Jordan, E., Sinha, V., Jackson, D.: Using dependency models to manage complex software architecture. SIGPLAN Not. **40** (2005) 167–176
20. Canfora, G., Penta, M.D.: Testing services and service-centric systems: Challenges and opportunities. IT Professional **8** (2006) 10–17
21. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: End-to-end versioning support for web services. Services Computing, 2008. SCC '08. IEEE International Conference on **1** (2008) 59–66
22. Luqi: A graph model for software evolution. IEEE Transactions on Software Engineering **16** (1990) 917–927
23. Michlmayr, A., Rosenberg, F., Leitner, P., Dustdar, S.: Advanced event processing and notifications in service runtime environments. In: DEBS. (2008) 115–125