Delft University of Technology
Software Engineering Research Group
Technical Report Series

Analyzing the Evolution of Web Services using Fine-Grained Changes

Daniele Romano and Martin Pinzger

Report TUD-SERG-2012-011

TUD-SERG-2012-011

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: Accepted for publication in the Proceedings of the International Conference on Web Services, 2012, IEEE CS Press.

Analyzing the Evolution of Web Services using Fine-Grained Changes

Daniele Romano
Software Engineering Research Group
Delft University of Technology
Delft, The Netherlands
Email: daniele.romano@tudelft.nl

Martin Pinzger
Software Engineering Research Group
Delft University of Technology
Delft, The Netherlands
Email: m.pinzger@tudelft.nl

***Abstract*—In the service-oriented paradigm web service interfaces are considered contracts between web service subscribers and providers. However, these interfaces are continuously evolving over time to satisfy changes in the requirements and to fix bugs. Changes in a web service interface typically affect the systems of its subscribers. Therefore, it is essential for subscribers to recognize which types of changes occur in a web service interface in order to analyze the impact on his/her systems.**

In this paper we propose a tool called *WSDLDiff* to extract fine-grained changes from subsequent versions of a web service interface defined in WSDL. In contrast to existing approaches, *WSDLDiff* takes into account the syntax of WSDL and extracts the WSDL elements affected by changes and the types of changes. With *WSDLDiff* we performed a study aimed at analyzing the evolution of web services using the fine-grained changes extracted from the subsequent versions of four real world WSDL interfaces.

The results of our study show that the analysis of the fine-grained changes helps web service subscribers to highlight the most frequent types of changes affecting a WSDL interface. This information can be relevant for web service subscribers who want to assess the risk associated to the usage of web services and to subscribe to the most stable ones.

***Keywords*-SOA; web services; software evolution; fine-grained changes;**

I. Introduction

Over the last decades, the evolution of software systems has been studied in order to analyze and enhance the software development and maintenance processes. Among other applications, the information mined from the evolution of software systems has been applied to investigate the causes of changes in software components [13] [7] [10]. Software engineering researchers have developed several tools to extract information about changes from software artifacts [5] [16] [19] and to analyze their evolution.

In service-oriented systems understanding and coping with changes is even more critical and challenging because of the distributed and dynamic nature of services [9]. In fact, service providers do not necessarily know the service subscribers and how changes on a service can impact the existing service clients. For this reason service interfaces are considered contracts between providers and subscribers and they should be as stable as possible [3]. On the other hand, services are continuously evolving to satisfy changes in the requirements and to fix bugs. Recognizing the types of changes is fundamental for understanding how a service interface evolves over time. This can help service subscribers to quantify the risk associated to the usage of a particular service and to compare the evolution of different services with similar features. Moreover, detailed information about changes allow software engineering researchers to analyze the causes of changes in a service interface.

In order to analyze the evolution of WSDL[1] interfaces, Fokaefs *et al.* [6] propose a tool called VTracker. This tool is based on the Zhang-Shashas tree-edit distance [20] comparing WSDL interfaces as XML[2] documents. However, VTracker does not take into account the syntax of WSDL interfaces. As consequence, their approach outputs only the percentage of added, changed and removed XML elements. We argue that this information is inadequate to analyze the evolution of WSDL interfaces without manually checking the types of changes and the WSDL elements affected by changes. Moreover, their approach of transforming a WSDL interface into a simplified representation can lead to the detection of multiple changes while there has been only one change.

In this paper we propose a tool called *WSDLDiff* that compares subsequent versions of WSDL interfaces to automatically extract the changes. In contrast to VTracker, *WSDLDiff* takes into account the syntax of WSDL and XSD,[3] used to define data types in a WSDL interface. In particular, *WSDLDiff* extracts the types of the elements affected by changes (*e.g.*, *Operation*, *Message*, *XSDType*) and the types of changes (*e.g.*, removal, addition, move, attribute value update). We refer to these changes as fine-grained changes. The fine-grained changes extraction process of *WSDLDiff* is based on the UMLDiff algorithm [19] and has been implemented on top of the *Eclipse Modeling Framework (EMF)*.[4]

With *WSDLDiff* we performed a study aimed at analyzing the evolution of web services using the fine-grained changes

[1] http://www.w3.org/TR/wsdl
[2] http://www.w3.org/XML/
[3] http://www.w3.org/XML/Schema
[4] http://www.eclipse.org/modeling/emf/

extracted from subsequent versions of four real world WSDL interfaces. We address the following two research questions:

- **RQ1:** What is the percentage of added, changed and removed elements of a WSDL interface?
- **RQ2:** Which types of changes are made to the elements of a WSDL interface?

The study shows that different WSDL interfaces are affected by different types of changes highlighting how they are maintained with different strategies. While in one case mainly *Operations* were added continuously, in the other three cases the data type specifications were the most affected by changes. Moreover, we found that in all four WSDL interfaces under analysis there is a type of change that is predominant. From this information web service subscribers can be aware of the frequent types of changes when subscribing to a web service and they can compare the evolution of web services that provide similar features in order to subscribe to the most stable web service.

The remainder of this paper is organized as follows. In Section II we report the related work and we discuss the main differences with our work. Section III describes the *WSDLDiff* tool and the process to extract fine-grained changes implemented into it. The study and results are presented in Section IV. We draw our conclusions and outline directions for future work in Section V.

II. Related Work

The most recent work on web services evolution has been developed by Fokaefs *et al.* [6] in 2011. They analyzed the evolution of web services using a tool called VTracker. This tool is based on the Zhang-Shasha's tree edit distance algorithm [20], which calculates the minimum edit distance between two trees. In this study the WSDL interfaces are compared as XML files. Specifically the authors created an intermediate XML representation to reduce the verbosity of the WSDL specification. In this simplified XML representation, among other transformations, the authors trace the references between messages parameters (*Parts*) and data types (*XSDTypes*) and they replace the references with the data types themselves. The output of their analysis consists of the percentage of added, changed and removed elements among the XML models of two WSDL interfaces. There are two main differences between our work and the approach proposed by Fokaefs *et al.* First, we compute the changes between WSDL models taking into account the syntax of WSDL and XSD and, hence, extracting the type of the elements affected by changes (e.g., *Operation*, *Message*, *XSDType*) and the types of changes (e.g., removal, addition, move, attribute value update). For example, *WSDLDiff* extracts differences in the order of the elements only if it is relevant, such as changes in the order of *Parts* defined in a *Message*. Our approach is aware of irrelevant order changes, such as changes in the order of *XSDTypes* defined in the WSDL types definition. This allows us to analyze the evolution of a WSDL interface only looking at the changes without manually inspecting the XML coarse-grained changes. Second, *WSDLDiff* does not replace the references to data types with the data types themselves. This transformation can lead to the detection of a change in a data type multiple times while there has been only one change.

In 2009 Wang *et al.* [17] proposed an impact analysis model based on service dependency. The authors analyze the service dependencies graph model, service dependencies and the relation matrix. Based on this information they infer the impact of the service evolution. However, they do not propose any technique to analyze the evolution of web services. In 2005 Aversano *et al.* [1] proposed an approach to understand how relationships between sets of services change across service evolution. Their approach is based on formal concept analysis. They used the concept lattice to highlight hierarchy relationships and to identify commonalities and differences between services. While the work proposed by Aversano *et al.* consists in extracting relationships among services, our work focuses on the evolution of single web services using fine-grained changes. As future work the two approaches can be integrated to correlate different types of changes with the different relationships.

In literature several approaches have been proposed to measure the similarity of web services (*e.g.*, [8] [12]). However, these approaches compute the similarity amongst WSDL interfaces to assist the search and classification of web services and not to analyze their evolution.

Concerning the model differencing techniques, the approach proposed by Xing *et al.* [19] [18] is most relevant for our work. In fact, their algorithm to infer differences among UML[5] diagrams has been implemented by the *EMF Compare*[6] that we used to implement our tool *WSDLDiff*. The authors proposed the *UMLDiff* algorithm for detecting structural changes between the designs of subsequent versions of object oriented systems, represented through UML diagrams. This algorithm has been later adapted in the *EMF Compare* to compare models conforming to any arbitrary metamodel and not only UML models [2].

Several approaches have been proposed to classify changes in service interfaces. For instance Feng *et al.* [4] and Treiber *et al.* [15] have proposed approaches to classify the changes of web services taking into account their impact to different stakeholders. These classifications can be easily integrated in our tool to classify the different fine-grained changes extracted along the evolution of a web service.

As can be deduced from the overview of related work there currently does not exist any tool for extracting fine-grained changes amongst web services. In this paper, we present such a tool based on the *UMLDiff* algorithm [19].

[5] http://www.uml.org/
[6] http://www.eclipse.org/emf/compare/

III. WSDLDIFF

In this section, we illustrate the *WSDLDiff* tool used to extract the fine-grained changes between two versions of a WSDL interface. Since the tool is based on the *Eclipse Modeling Framework*, we first present an overview of this framework and then we describe the fine-grained changes extraction process implemented by *WSDLDiff*. A first prototype of *WSDLDiff* is available on our web site.[7]

A. Eclipse Modeling Framework

The *Eclipse Modeling Framework (EMF)* is a modeling framework that lets developers build tools and other applications based on a structured data model. This framework provides tools to produce a set of Java classes from a model specification and a set of adapter classes that enable viewing and editing of the models. The models are described by meta models called *Ecore*.

As part of the *EMF* project, there is the *EMF Compare* plug-in. It provides comparison and merge facilities for any kind of *EMF Models* through a framework easy to be used and extended to compare instances of *EMF Models*. The Eclipse community provides already an *Ecore* meta model for WSDL interfaces, including a meta model for XSD, and tools to parse them into *EMF Models*. We use these features to parse and extract changes between WSDL interfaces as described in the following.

B. Fine-Grained Changes Extraction Process

Figure 1 shows the process implemented by *WSDLDiff* to extract fine-grained changes between two versions of a WSDL interface. The process consists of four stages:

- **Stage A:** in the first stage we parse the WSDL interfaces using the APIs provided by the *org.eclipse.wst.wsdl* and *org.eclipse.xsd* projects. The output of this stage consists of the two *EMF Models* (*WSDL Model1* and *WSDL Model2*) corresponding to the two WSDL interfaces taken as input (*WSDL Version1* and *WSDL Version2*).
- **Stage B:** in this stage we transform the *EMF Models* corresponding to the XSD (contained by the WSDL models) in order to improve the accuracy of the fine-grained changes extraction process as it will be shown in the Subsection III-D. The output of this stage consist of the transformed models (*WSDL Model1'* and *WSDL Model2'*).
- **Stage C:** in the third stage we use the *Matching Engine* provided by the *EMF Compare* framework to detect the nodes that match in the two models.
- **Stage D:** the *Match Model* produced by the *Matching Engine* is then used to detect the differences among the two WSDL models under analysis. This task is accomplished by the *Differencing Engine* provided also by *EMF Compare*. The output of this stage is a tree of structural changes that reports the differences between the two WSDL models. The differences are reported in terms of additions, removals, moves and modifications of each element specified in the WSDL and in the XSD.

[7] http://swerl.tudelft.nl/twiki/pub/DanieleRomano/WebHome/WSDLDiff.zip

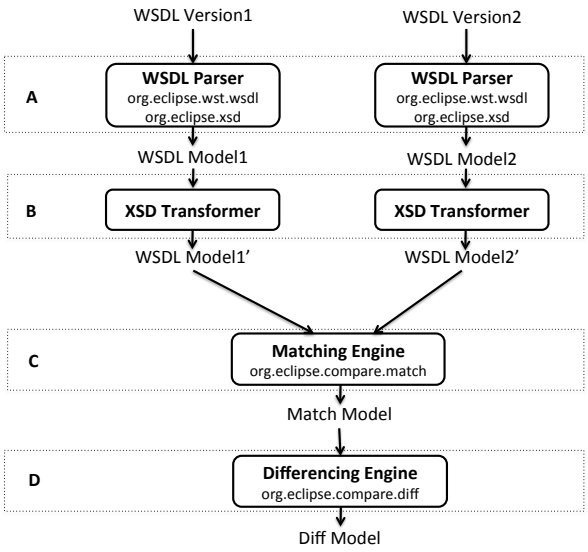


Figure 1: The process implemented by *WSDLDiff* to extract fine-grained changes between two versions of a WSDL interface.

In the next subsection we first illustrate the strategies behind *EMF Compare* describing the matching (Stage C) and differencing (Stage D) stages and then we describe the XSD transformation (Stage B).

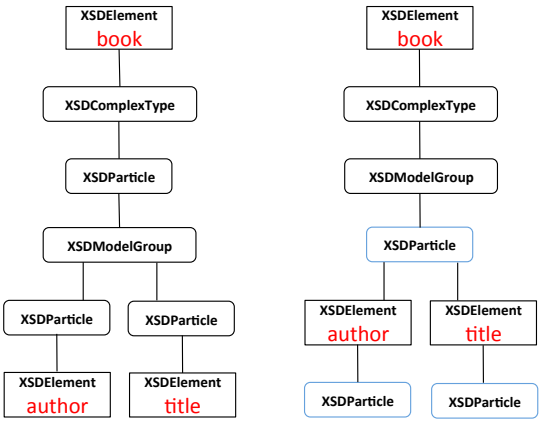
C. Eclipse EMF Compare

The comparison facility provided by *EMF Compare* is based on the work developed by Xing *et al.* [19]. This work has been adapted to compare generic *EMF Models* instead of UML models as initially developed by Xing. The comparison consists of two phases: (1) the matching phase (Stage C in our approach) and (2) the differencing phase (Stage D in our approach). The matching phase is performed computing a set of similarity metrics. These metrics are computed for two nodes while traversing the two models under analysis with a top-down approach. In the generic *Matching Engine*, provided in *org.eclipse.compare.match* and used in our approach, the set of metrics consists of four similarity metrics:

- **type similarity:** to compute the match of the types of two nodes;
- **name similarity:** to compute the similarity between the values of the attribute *name* of two nodes;

```
<xs:element name="book">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="author" type="xs:string"/>
   <xs:element name="title" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

(a) Definition of an XSD element



(b) Original EMF Model (c) Transformed EMF Model

Figure 2: An example that shows the XSD transformation performed by the *XSD Transformer* in the Stage B of the fine-grained changes extraction process.

- **value similarity:** to compute the similarity between the values of other attributes declared in the nodes;
- **relations similarity:** to compute the similarity of two nodes based on the relationships they have with other nodes (*e.g.*, children and parents in the model).

Once the matching phase has been completed, it produces a matching model consisting of all the entities that are matched in the two models. The matching model is then used in the differencing phase to extract all the differences between the two models. Specifically, the matching model is browsed by a *Differencing Engine* that computes the tree edit operations. These operations represent the minimum set of operations to transform a model into an other model. They are classified in *added*, *changed*, *removed* and *moved* operations. For more details about the matching and differencing phases implemented by *EMF Compare* we refer the reader to [2].

D. XSD Transformation

In an initial manual validation of *EMF Compare* on WSDL models we found that in a particular case the set of differences produced did not correspond to the minimum set of tree edit operations. The problem was due to the *EMF Model* used to represent the XSDs. For this reason we decided to add the *XSD Transformer*. To better understand the problem behind the original *EMF Model* and the solution adopted, consider the example shown in Figure 2. Figure 2a shows an *XSDElement book* that consists of an *XSDModelGroup* (the element *sequence*) that contains two *XSDElements* (the elements *author* and *title*). Figure 2b shows the original *EMF Model* parsed by the *WSDL Parser* (Stage A in Figure 1). The *EMF Model* contains the nodes *XSDParticle*. These nodes are necessary to represent the attributes *minOccurs*, *maxOccurs* and *ref* for each *XSDElement* declared in an *XSDModelGroup* and for the *XSDModelGroup* itself.

The *XSDParticles* in the original model are parents of the elements to which they are associated. This structure can lead to mistakes when the order of *XSDElements* within an *XSDModelGroup* changes. In this case, when the *Matching Engine* traverses the models, it can detect a match between *XSDParticles* that are associated to different *XSDElements* (*e.g.*, a match between the *XSDParticle* of the element *author* and the *XSDParticle* of the element *title*). This match is likely because the values of the attributes *minOccurs*, *maxOccurs* and *ref* are set to their default values. When this match occurs the *Matching Engine* keeps traversing the model and it detects a mismatch when it traverses the children of the previously matched *XSDParticles* (*e.g.*, a mismatch between the elements *author* and *title*). As consequence, even if there are no differences among the models the *Differencing Engine* can produce the *added XSDelement title*, the *added XSDelement author*, the *removed XSDelement title* and the *removed XSDelement author* as changes.

To overcome this problem, we decided to transform the *EMF Model* inverting the parent-child relationship in presence of *XSDParticles* as shown in Figure 2c. In the transformed models, the *Matching Engine* traverses the *XSDParticles* only when a match is detected between the *XSDElements* to which they are associated.

Besides this problem, in one case, *WSDLDiff* reported the *removed Part* and *added Part* changes instead of the *changed Part* change when a *Part* was renamed. However for this study the two set of changes are equivalent. For this reason we have not considered it as a problem. Clearly, as part of our future work we plan to validate the fine-grained changes extraction process with a benchmark.

IV. Study

The *goal* of this study is to analyze the evolution of web services through the analysis of fine-grained changes extracted from subsequent versions of a WSDL interface. The *perspective* is that of web services subscribers interested in extracting the types of changes that appear along the evolution of a web service. They can analyze the most frequent changes in a WSDL interface estimating the risk related to the usage of a specific element. The *context* of this

study consists of all the publicly available WSDL versions of four real world web services, namely:

- **Amazon EC2:** Amazon Elastic Compute Cloud is a web service that provides resizable compute capacity in the cloud. In this study we have analyzed 22 versions.
- **FedEx Rate Service:** the Rate Service provides the shipping rate quote for a specific service combination depending on the origin and destination information supplied in the request. We analyzed 10 different versions.
- **FedEx Ship Service:** the Ship Service provides functionalities for managing package shipments and their options. 7 versions out of 10 have been analyzed in this study.
- **FedEx Package Movement Information Service:** the Package Movement Information Service provides operations to check service availability, route and postal codes between origin and destination. We analyzed 3 versions out of 4. For the sake of simplicity we refer to this service as *FedEx Pkg*.

We chose these web services because they were previously used by Fokaefs *et al.* [6]. The other web services analyzed by Fokaefs *et al.* [6] (*PayPal SOAP API*[8] and *Bing Search*[9]) have not been considered because the previous versions of the WSDL interfaces are not publicly available. For the same reasons not every version of the web services has been considered in our analysis. In Table I we report the size of the WSDL interfaces in terms of number of *Operations*, number of *Parts*, number of *XSDElements* and number of *XSDTypes* declared in each version. The size of the WSDL interfaces has been measured using the API provided by the *org.eclipse.wst.wsdl* and *org.eclipse.xsd* Eclipse Plug-in projects.

The results reported in Table I show that the web services under analysis evolve differently. The number of *Operations* declared in the *AmazonEC2* service is continuously growing and only in four versions does not change (version 5, 7, 22 and 23). The number of *Operations* declared in the other web services is more stable. Specifically, the *FedEx Pkg* service declares always 2 *Operations*. The *FedEx Rate* service declares 1 *Operation* in 9 versions out of 10 and 2 *Operations* in 1 version (version 3). Concerning the *FedEx Ship* service we can notice an increase in the number of *Operations* from version 1 to version 5. Then, the number of *Operations* decreases to 7 and it remains stable until the current version (version 10).

To better understand the evolution of web services we used the *WSDLDiff* tool to extract the fine-grained changes from subsequent versions of the WSDL interfaces under analysis. In the next subsections we first show the types of changes extracted in this study and then we present the results of the study answering our research questions.

[8]https://www.paypalobjects.com/enUS/ebook/PPAPIReference/architecture.html

[9]http://www.bing.com/developers

Table I: Number of *Operations*, *Parts*, *XSDElements* and *XSDTypes* declared in each version of the WSDL interfaces under analysis

WSDL	Ver.	Operations	Parts	XSDElements	XSDTypes
AmazonEC2	2	14	28	28	60
AmazonEC2	3	17	34	34	75
AmazonEC2	4	19	38	38	81
AmazonEC2	5	19	38	38	81
AmazonEC2	6	20	40	40	87
AmazonEC2	7	20	40	40	85
AmazonEC2	8	26	52	52	111
AmazonEC2	9	34	68	68	137
AmazonEC2	10	37	74	74	151
AmazonEC2	11	38	76	76	157
AmazonEC2	12	41	82	82	171
AmazonEC2	13	43	86	86	179
AmazonEC2	14	65	130	130	259
AmazonEC2	15	68	136	136	272
AmazonEC2	16	74	148	148	296
AmazonEC2	17	81	162	162	326
AmazonEC2	18	87	174	174	350
AmazonEC2	19	91	182	182	366
AmazonEC2	20	95	190	190	390
AmazonEC2	21	118	236	236	464
AmazonEC2	22	118	236	236	465
AmazonEC2	23	118	236	236	467
FedEx Rate	1	1	2	2	72
FedEx Rate	2	1	2	2	80
FedEx Rate	3	2	4	4	88
FedEx Rate	4	1	2	2	124
FedEx Rate	5	1	2	2	129
FedEx Rate	6	1	2	2	178
FedEx Rate	7	1	2	2	202
FedEx Rate	8	1	2	2	223
FedEx Rate	9	1	2	2	228
FedEx Rate	10	1	2	2	235
FedEx Ship	2	1	2	2	124
FedEx Ship	5	9	16	16	178
FedEx Ship	6	9	16	16	177
FedEx Ship	7	7	12	12	199
FedEx Ship	8	7	12	12	221
FedEx Ship	9	7	12	12	246
FedEx Ship	10	7	12	12	254
FedEx Pkg	2	2	4	4	20
FedEx Pkg	3	2	4	4	20
FedEx Pkg	4	2	4	4	20

A. Fine-Grained Changes

The output of *WSDLDiff* consists of the set of edit operations. These operations are associated with the elements declared in the WSDL and XSD specifications. Among all the elements the following WSDL elements have been detected as affected by changes: *BindingOperation*, *Operation*, *Message* and *Part*. The XSD elements detected as affected by changes are: *XSDType*, *XSDElement*, *XSDAttributeGroup* and *XSDAnnotation*. These elements were affected by the following fine-grained changes:

- **XSD Element changes:** consist of added *XSDElements* (*XSDElementA*), removed *XSDElements* (*XS-*

DElementR) and moved *XSDElements* (*XSDElementM*) within a declaration of an *XSDType* or an *XSDElement*.
- **Attribute changes:** changes due to the update of an attribute value. Specifically we detected changes to the values of the attributes name (*NameUpdate*), minOccurs (*MinOccursUpdate*), maxOccurs (*MaxOccursUpdate*) and fixed (*FixedUpdate*).
- **Reference Changes:** consists of changes to a referenced value (*RefUpdate*).
- **Enumeration Changes:** changes of elements declared within an *XSDEnumeration* element. We detected added enumeration values (*EnumerationA*) and removed enumeration values (*EnumerationR*).

For the sake of simplicity we have presented only the changes detected in our study. However *WSDLDiff* is able to detect changes to every element declared in the WSDL and XSD specifications.

B. Research Question 1 (RQ1)

The first research question (RQ1) is:

What is the percentage of added, changed and removed elements of a WSDL interface?

To answer RQ1, for each type of element declared in the WSDL and XSD specifications, we counted the number of times they have been added, changed, or removed between every pair of subsequent versions of the WSDL interfaces under analysis. We present the results in three different tables. In Table II we report the number of added, changed and deleted WSDL elements while the added, changed and removed XSD elements are shown in Table III. Table IV summarizes the results showing the total number and the percentage of added, changed and deleted WSDL and XSD elements for each web service. The raw data with the changes extracted for each pair of subsequent versions is available on our web site.[10] In Table II we omitted the number of added, changed and removed *BindingOperations* because they are identical to the number of added, changed and removed *Operations*. Moreover, the added and removed *Parts* do not include the *Parts* that were added and removed due to the additions and deletions of *Messages*. This choice allows us to highlight the changes in the *Parts* of existing *Messages*.

The results show that in all the web services the total number of deleted elements is a small percentage of the total number of changes (see Table IV). In particular, the percentage of deleted elements is approximately 4% for AmazonEC2, 12% for FedEx Rate and 6% for FedEx Ship. This result demonstrates that web service providers do not tend to delete existing elements in order to avoid breaking their clients.

[10] http://swerl.tudelft.nl/twiki/pub/DanieleRomano/WebHome/ICWS12RQ1.pdf

Table II: Number of added *Operations* (*OperationA*), changed *Operations* (*OperationC*), deleted *Operations* (*OperationD*), added *Messages* (*MessageA*), changed *Messages* (*MessageC*), deleted *Messages* (*MessageD*), added *Parts* (*PartA*), changed *Parts* (*PartC*) and deleted *Parts* (*PartD*) for each WSDL interface.

Change Type	AmazonEC2	FedEx Rate	FedEx Ship	FedEx Pkg
OperationA	113	1	10	0
OperationC	0	1	0	0
OperationD	9	1	4	0
MessageA	218	2	16	0
MessageC	2	0	2	0
MessageD	10	2	2	0
PartA	27	0	2	0
PartC	34	0	0	0
PartD	27	0	2	0
Total	**440**	**7**	**38**	**0**

Table III: Number of added *XSDTypes* (*XSDTypeA*), changed *XSDTypes* (*XSDTypeC*), deleted *XSDTypes* (*XSDTypeD*), added *XSDElements* (*XSDElementA*), changed *XSDElements* (*XSDElementC*), deleted *XSDElements* (*XSDElementD*), added *XSDAttributeGroup* (*XSDAttributeGroupA*) and changed *XSDAttributeGroup* (*XSDAttributeGroupC*) for each WSDL interface.

Change Type	AmazonEC2	FedEx Rate	FedEx Ship	FedEx Pkg
XSDTypeA	409	234	157	0
XSDTypeC	160	295	280	6
XSDTypeD	2	71	28	0
XSDElementA	208	2	25	0
XSDElementC	1	0	18	0
XSDElementD	0	2	0	0
XSDAttributeGroupA	6	0	0	0
XSDAttributeGroupC	5	0	0	0
Total	**791**	**604**	**508**	**6**

Concerning the number of added elements, the FedEx Rate and Ship services show approximately the same percentage (39% and 38%) while the AmazonEC2 service shows a percentage of approximately 80%. These percentages need to be interpreted taking into account the added, changed and removed WSDL and XSD elements. In fact, while the AmazonEC2 evolves continuously adding 113 new *Operations* (see Table II), the FedEx services are more stable with 1 new *Operation* added in FedEx Rate and 10 new *Operations* added in FedEx Ship. However, despite the few number of new *Operations* added in the FedEx services the number of added, changed and removed *XSDTypes* is high like in the AmazonEC2 service. This result lets us assume that the elements added in the FedEx services modify old functionalities and, hence, they are more likely to break the clients. Instead the AmazonEC2 is continuously evolving providing new *Operations*. This assumption is confirmed by the percentage of changed elements, that is lower in AmazonEC2 (about 16%) than in FedEx Rate and Ship

Table IV: Number of added, changed and removed WSDL and XSD elements for each WSDL interface under analysis

WSDL	Type	#Added	#Changed	#Deleted
AmazonEC2	WSDL	358	34	46
AmazonEC2	XSD	623	166	5
AmazonEC2	**Total**	**981 (≈80%)**	**200 (≈16%)**	**51 (≈4%)**
FedEx Rate	WSDL	3	1	3
FedEx Rate	XSD	236	295	73
FedEx Rate	**Total**	**239 (≈39%)**	**296 (≈49%)**	**76 (≈12%)**
FedEx Ship	WSDL	28	4	8
FedEx Ship	XSD	182	298	28
FedEx Ship	**Total**	**210 (≈38%)**	**302 (≈55%)**	**36 (≈6%)**
FedEx Pkg	WSDL	0	0	0
FedEx Pkg	XSD	0	6	0
FedEx Pkg	**Total**	**0 (0%)**	**6 (100%)**	**0 (0%)**

(about 49% and 55%).

Based on these results we can answer RQ1 stating that in all four web services the percentage of removed elements is a small percentage compared to the total number of added, changed and removed elements. Concerning the added elements the AmazonEC2 showed the highest percentage (≈80%) due to the high number of new WSDL elements added along its evolution. Instead the FedEx Rate and Ship services showed lower percentages (respectively about 39% and 38%). The percentage of changed elements is higher in the FedEx Rate and Ship services (respectively about 49% and 55%) compared to the approximately 16% of changed elements in AmazonEC2.

Answering RQ1 we decided to omit the analysis of the FedEx Pkg service because the low number of changes and versions do not allow us to make any assumption.

C. Research Question 2 (RQ2)

The second research question (RQ2) is:

Which types of changes are made to the elements of a WSDL interface?

In order to address RQ2 we focused on the changes applied to *XSDTypes*. In fact, among all the elements changed (802), 742 elements (approximately 92%) are *XSDTypes* (see Table II and III). For each *XSDType* we extracted the fine-grained changes and we report the results in Table V. We omitted to report the number of *XSDAnnotation* changes because they are not relevant for our study. The raw data with the changes extracted for each pair of subsequent versions is available on our web site.[11]

The results show that the most frequent change along the evolution of the *AmazonEC2* is the *XSDElementA*. In fact, it accounts for around 80% (198 changes out of 247) of the total changes. Concerning the *FedEx Rate* and *FedEx Ship* services, the *EnumerationA* changes are the most

[11] http://swerl.tudelft.nl/twiki/pub/DanieleRomano/WebHome/ICWS12RQ2.pdf

Table V: Number of added *XSDElements* (*XSDElementA*), deleted *XSDElements* (*XSDElementR*), moved *XSDElements* (*XSDElementM*), updated attributes (*NameUpdate*, *MinOccursUpdate*, *MaxOccursUpdate* and *FixedUpdate*), updated references (*RefUpdate*), added enumeration values (*EnumerationA*) and removed enumeration values (*EnumerationR*) in the *XSDTypes* for each WSDL interface.

Change Type	AmazonEC2	FedEx Rate	FedEx Ship	FedEx Pkg
XSDElementA	**198**	113	136	1
XSDElementD	11	47	49	**3**
XSDElementM	1	55	51	0
NameUpdate	11	20	8	0
MinOccursUpdate	17	33	39	0
MaxOccursUpdate	0	9	6	0
FixedValue	0	11	12	2
RefUpdate	9	80	273	0
EnumerationA	0	**1141**	**926**	2
EnumerationD	0	702	528	**3**
Total	**247**	**2211**	**2028**	**11**

frequent, accounting for approximately 51% (1141 changes out of 2211) and for 45% (926 changes out of 2028) of all changes. Adding the *EnumerationD* changes, we obtain approximately 83% (1843 changes out of 2211) and 71% (1454 changes out of 2028) of changes occurring in the enumeration elements. The results show that in 3 web services out of 4 there is a type of change that is predominant. This result demonstrates that fine-grained changes can help web services subscribers to be aware of the most frequent types of changes affecting a WSDL interface. Like for RQ1, the small number of changes in the *FedEx Pkg* does not allow any valid conclusion.

D. Summary and implications of the results

The changes collected in this study highlight how different WSDL interfaces evolve differently. This study with the *WSDLDiff* tool can help services subscribers to analyze which elements are frequently added, changed and removed and which types of changes are performed more frequently. For example, a developer who wants to integrate a *FedEx* service into his/her application can learn that the specification of data types changes most frequently while *Operations* change only rarely (RQ1). In particular, the enumeration values are the most unstable elements (RQ2). Instead, an *AmazonEC2* subscriber can be aware that new *Operations* are continuously added (RQ1) and that data types are continuously modified adding new elements (RQ2).

V. Conclusion & Future Work

In this paper we proposed a tool called *WSDLDiff* to extract fine-grained changes between two WSDL interfaces. With *WSDLDiff* we performed a study aimed at understanding the evolution of web services looking at the changes detected by our tool. The results of our study showed that the fine-grained changes are a useful means to understand how a

particular web service evolves over time. This information is relevant for web services subscribers who want 1) to analyze the most frequent changes affecting a WSDL interface and 2) to compare the evolution of different web services with similar features. From this information they can estimate the risk associated to the usage of a web service.

The study presented in this paper is the first study on the evolution of web services and we believe that our tool provides an essential starting point. As future work, first we plan to classify the changes retrievable with *WSDLDiff*, integrating and possibly extending the works proposed by Feng *et al.* [4] and Treiber *et al.* [15]. Next, we plan to investigate metrics that can be used as indicators of changes in WSDL elements. For instance in our previous work [13], we found an interesting correlation between the number of changes in Java interfaces and the external cohesion metric defined for services by Perepletchikov *et al.* [11]. With our tool to extract fine-grained changes and our previous work to extract dependencies among web services [14] we plan to perform similar studies with WSDL interfaces. Finally, we plan to investigate the co-evolution of the different web services composing a service oriented system. With *WSDLDiff* we can highlight web services that evolve together, hence, violating the *loosely coupling* property. This analysis can help us to investigate the causes of web services co-evolution and techniques to keep their evolution independent.

ACKNOWLEDGMENT

This work has been partially funded by the NWO-Jacquard program within the ReSOS project.

REFERENCES

[1] L. Aversano, M. Bruno, M. D. Penta, A. Falanga, and R. Scognamiglio. Visualizing the evolution of web services using formal concept analysis. In *IWPSE*, pages 57–60, 2005.

[2] C. Brun and A. Pierantonio. Model differences in the eclipse modelling framework. *UPGRADE The European Journal for the Informatics Professional*, IX(2):29–34, 2008.

[3] T. Erl. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

[4] Z. Feng, K. He, R. Peng, and Y. Ma. Taxonomy for evolution of service-based system. In *SERVICES*, pages 331–338, 2011.

[5] B. Fluri, M. Wuersch, M. PInzger, and H. Gall. Change distilling: Tree differencing for fine-grained source code change extraction. *IEEE Trans. Softw. Eng.*, 33:725–743, November 2007.

[6] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau. An empirical study on web service evolution. In *ICWS*, pages 49–56, 2011.

[7] F. Khomh, M. Di Penta, and Y.-G. Gueheneuc. An exploratory study of the impact of code smells on software change-proneness. In *Proceedings of the 2009 16th Working Conference on Reverse Engineering*, WCRE '09, pages 75–84, Washington, DC, USA, 2009. IEEE Computer Society.

[8] F. Liu, Y. Shi, J. Yu, T. Wang, and J. Wu. Measuring similarity of web services based on wsdl. In *ICWS*, pages 155–162, 2010.

[9] M. P. Papazoglou. The challenges of service evolution. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, CAiSE '08, pages 1–15. Springer-Verlag, 2008.

[10] M. D. Penta, L. Cerulo, Y.-G. Guéhéneuc, and G. Antoniol. An empirical study of the relationships between design pattern roles and class change proneness. In *ICSM*, pages 217–226, 2008.

[11] M. Perepletchikov and C. Ryan. The impact of service cohesion on the analyzability of service-oriented software. *IEEE T. on Software Engineering*, 37(4):449–465, 2011.

[12] P. Plebani and B. Pernici. Urbe: Web service retrieval based on similarity evaluation. *IEEE Trans. on Knowl. and Data Eng.*, 21:1629–1642, November 2009.

[13] D. Romano and M. Pinzger. Using source code metrics to predict change-prone java interfaces. In *ICSM*, pages 303–312, 2011.

[14] D. Romano, M. Pinzger, and E. Bouwers. Extracting dynamic dependencies between web services using vector clocks. In *SOCA*, pages –, 2011.

[15] M. Treiber, H. L. Truong, and S. Dustdar. On analyzing evolutionary changes of web services. In *ICSOC Workshops*, pages 284–297, 2008.

[16] N. Tsantalis, N. Negara, and E. Stroulia. Webdiff: A generic differencing service for software artifacts. In *ICSM*, pages 586–589, 2011.

[17] S. Wang and M. A. M. Capretz. A dependency impact analysis model for web services evolution. In *ICWS*, pages 359–365, 2009.

[18] Z. Xing and E. Stroulia. Analyzing the evolutionary history of the logical design of object-oriented software. *IEEE Trans. Software Eng.*, 31(10):850–868, 2005.

[19] Z. Xing and E. Stroulia. Umldiff: an algorithm for object-oriented design differencing. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 54–65, 2005.

[20] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18:1245–1262, December 1989.

TUD-SERG-2012-011
ISSN 1872-5392