WSDarwin: A Comprehensive Framework for Supporting Service-Oriented Systems Evolution

by

Marios-Eleftherios Fokaefs

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

Service-oriented architecture (SOA) has become the prevalent paradigm for the development of distributed and modular software systems. SOA owes its popularity to certain properties that characterize the resulting systems and, in theory, gives flexibility to the development and maintenance of service-oriented systems. First, web services, which are the building blocks of service-oriented systems, are accessible over the Internet, which implies that there is no need to exchange code artifacts (as in the case of software libraries). Second, software components in service-oriented systems are published through concise and abstract interfaces usually based on common and well-adopted standards. The abstraction implies that the interface exposes just enough for a functional run-time data exchange. The abstraction results in information hiding, which offers two benefits. On one hand, providers can hide the business logic of their service from the clients, thus, retaining their expertise or the ownership of data, from which they can possibly profit. On the other hand, taking advantage of the common standards, clients can flexibly migrate between web services to satisfy their requirements as best as possible.

Nevertheless, these particular properties may also cause problems in the stability of a service-oriented system especially in the case of software evolution, if good practices are not followed. One relevant challenge is that, since the system is distributed, the decision about the evolution of a service may be restrained to a small part of the system. Also, since the communication between the components is limited by the abstract interface, the impact of the change may be unknown for the rest of the system. The clients of a service may have little information about the changes and how to react to them. The published service interface constitutes a contract between the provider and

the client. If best practices about service evolution are not followed and this contract is broken, the repercussions may be severe to the client application and this can also have business and financial impact both to the client and the provider. Another challenge is caused by the definition and availability of multiple styles and technologies that follow a service-oriented architecture, including, but not limited to, REST and WS-* services. This results in a variability on how service-oriented systems are implemented. Although the challenges around the evolution of web services may be based on fundamental properties of the architecture, each style may require different solutions and tools to support the maintenance of service-oriented systems.

Given these challenges, there is an evident need to support client developers in the maintenance of their application against evolving services. This support is necessary regardless of whether good practices are followed by providers and regardless of the underlying styles and technologies of the service system. There is also a need to support the decision-making processes of service providers and clients about the evolution of their software, in a manner aware not only of the technical but also the relevant business and economic considerations. In my work, I make three contributions towards satisfying these needs. I have developed the WSDarwin tools to support the maintenance of client applications in the event of service evolution; first, an Eclipse plug-in to support service clients of the WS-* style and, second, a web application to support REST service clients. Between the two implementations, WSDarwin offers support for a variety of tasks, including comparison of WSDL and WADL service interfaces to identify differences between versions of a service, automatic adaptation of WS-* client applications, automatic generation of WADL interfaces for REST services and automatic mapping of similar services from different providers. The third contribution of my work is the development of a theoretical framework to support the decision-making process concerning the evolution of a service-oriented system. The framework consists of an economic model and a game-theoretic model to take into consideration the economic repercussions of service evolution and the complicated interactions between providers and clients. The thesis of my work is that service evolution should

be not only technically but also socially and economically conscious through support from automated tools. WSDarwin implements a suite of relevant tools, and thus substantiates the thesis.

Preface

This thesis is written in a paper-based format. It consists of papers that have already been published or of papers that have been submitted, but have not yet been reviewed by the time this thesis was submitted. Each paper is copied in its entirety and its original format in a dedicated section in the corresponding chapter. For the published papers, I start each section with the corresponding reference to the paper, while for the submitted paper, I provide the partial reference to the paper and a note to specify that the paper has been submitted, but not yet accepted and the date of submission.

There are two of the published papers that had more authors, other than myself, my supervisor or a member of my supervisory committee. For the paper presented in Section 2.1 (Fokaefs et al., 2011), the second co-author developed the tool I used to study the web services, while the third co-author found the different versions for the various services and the relevant business announcements. I conducted the empirical study, drew the conclusions and wrote the biggest part of the paper. For the paper presented in Section 3.2, I developed the mapping methodology and its implementation into a tool, designed its evaluation on the selected APIs and wrote the biggest part of the paper. The first author of this paper conducted the experiments to evaluate the mapping methodology and helped me write the evaluation section of the paper. In this case, the order of the authors reflect the fact that this work was a requirement for a course of the first author, and not necessarily the contribution of each author.

Acknowledgements

I would like to thank Professors Hoover, Messinger, Miller and Martin for serving as members for my PhD committee, all the reviewers of my publications and all the colleagues I have met and discussed in various conferences and especially the CSER community, whose comments and advice helped to shape and guide this work towards the right direction.

I would also like to thank my colleagues in the SERL lab and the SSRG group for creating an excellent work environment with great atmosphere. A huge thanks goes to all of my friends and especially Angeliki Altani, John Dimopoulos, Natasa Tsantali, Nikos Vitzilaios and Maria Attarian (from afar) for helping me keep my sanity and reminding me that there is so much more in life.

I couldn't possibly just thank my parents, because everything they have done for me to support me and encourage me in my work is invaluable. I hope I made them proud. I would be remiss if I didn't acknowledge the support from my professors and mentors Professors Chatzigeorgiou and Tsantalis and, of course, my supervisor Professor Eleni Stroulia, who has not only supported this work, but has also generously provided me with the necessary skills to pursue my future endeavours. *"I owe my parents my life and my teacher my happiness"*.

Table of Contents

1 Introduction **1**
- 1.1 Service-Oriented Architectures . . . 1
- 1.2 Problem Definition . . . 2
- 1.3 Technical Background . . . 6
- 1.4 Thesis and Contributions . . . 8
- 1.5 Outline . . . 12

2 Support for WS-* Service Clients **15**
- 2.1 An empirical study on web service evolution . . . 21
- 2.2 WSMeta: a meta-model for web services to compare service interfaces . . . 31
- 2.3 WSDarwin: Studying the Evolution of Web Service Systems . 40
- 2.4 WSDarwin: automatic web service client adaptation. . . . 66
- 2.5 The WSDarwin Toolkit for Service-Client Evolution . . . 82

3 Support for REST Applications **87**
- 3.1 Developing and Maintaining REST Client Applications: The Tumblr Case Study . . . 90
- 3.2 Mapping the responses of RESTful services based on their values 101
- 3.3 WSDarwin: A Web Application for the Support of REST Service Evolution . . . 112

4 Service Evolution Economics **117**
- 4.1 Software Evolution in the Presence of Externalities: A Game-Theoretic Approach . . . 120
- 4.2 WSDarwin: A Decision-Support Tool for Web-Service Evolution 145
- 4.3 Software Evolution in Web-Service Ecosystems: A Game-Theoretic Model . . . 150

5 Conclusions and Future Work **165**
- 5.1 Summary and Conclusions . . . 165
- 5.2 Future Plans and Directions . . . 167

Bibliography **170**

Chapter 1

Introduction

1.1 Service-Oriented Architectures

Service-oriented architecture (SOA) has become the prevalent style for the development of modular systems. Systems implemented based on this architecture are distributed and their components communicate with each other over the Internet. SOA enables interoperability and flexibility in developing such distributed systems based on existing software components. The flexibility of the architecture successfully addresses the high volatility of such systems in terms of changing requirements (functional and non-functional) and the availability (or absence thereof) of services. Another goal of the architecture is to overcome the technical barriers imposed by the existence of multiple platforms, programming languages and hardware constraints. The flexibility, interoperability and technological independence of SOA are achieved by relying on open Internet standards that can be supported by most, if not by all, platforms and programming languages.

Web services are the building block of SOA and the term is used to describe a broad class of software components. As a result of this variety, there are a number of perceptions as to what is a web service and in turn there is a number of definitions, an overview of which is provided by Alonso et al. (2004). In the context of this work and throughout this document, I assume the definition provided by the UDDI consortium, which describes web services as *"self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces"* (UDDI Consortium, 2001). This definition, simple

yet comprehensive, emphasizes all the properties of web services that make this work interesting and its contributions important for the advancement of Web Services research.

The self-containment of web services implies that they are independently designed, implemented and maintained software components, which are fully functional. Self-containment also implies a notion of ownership and a clear definition of boundaries between the service and other components. The definition of web services as business application clearly points out their importance for firms as business assets with financial and economic properties. Web services enable firms to take advantage of and expose their expertise in a domain, their data or certain qualities they may possess, including reliability, authority and security. As business assets, they have to be protected and this is the reason that the business logic of a web service is hidden behind a carefully designed interface. The fact that this interface is based on web standards, which are supported by most platforms and programming languages, implies that they are directly consumable by a machine, which facilitates the development of automated tools to support the development and maintenance of service-oriented systems.

1.2 Problem Definition

Unlike other modular and distributed architectures, SOA has properties that may challenge certain software-engineering tasks. This work is specifically concerned with the task of software evolution, namely how web services evolve and how this evolution affects the maintenance of a service system and its components in general. This section lays out the specifics of these properties and how they manifest themselves during service evolution.

Jurisdiction: As independently developed software components, the web services of a service-oriented system may belong to different entities. As a result, the decision about a change may be made independently and lie outside the control of other participants in the system. There are no formal mechanisms to enforce that changes should take into account the impact on the rest

of the system. Furthermore, the composition of the system in terms of services and clients may not always be completely known. As a result, a change to a service may affect the stability of certain components or of the system as a whole. *Backwards compatibility*, which suggests that a change can be implemented so that it doesn't affect current client applications, cannot always be supported. One way of offering backwards compatibility is by keeping the older version available. However, maintaining multiple versions of a service can result in additional costs to support the necessary infrastructure (e.g. servers) and it can actually be cheaper for client applications to migrate to the new version. Therefore, it is the normal policy of service providers, after a grace period, to take down an older version and force their clients to migrate to the new version (Twitter, 2014; Tumblr, 2014). In some cases, supporting backwards compatibility, by including deprecated interfaces for example, may cause the quality of the service to deteriorate and implies additional maintenance costs.

Information hiding: While hiding the business logic of a web service enables the owner to protect his intellectual property, it deprives the clients of valuable information about the implementation of the service especially in the event of a change. Since the client application perceives only an abstraction of the service's functionality through its interface, it is hard to assess the specific details of a change. If the change affects the service interface, it will be easy for a client to recognize that there has been a change. However, if the change goes deeper in the functionality of the service, it will still have an impact on the client's behaviour. If the change affects the service's functionality but not its interface, then the change will be perceived by the client through a fault. A provider may give more information about the change, but it will more than often be in a format that it is not directly consumable by a machine. While this information will assist the client to adapt to the new version of the service, it does not facilitate the implementation of tools to support the automatic adaptation of client applications, which is one of the goals of SOA.

Live connection: Client applications need to maintain a live connection with web services, since the latter are web-accessible software. Unlike software libraries, where a client owns a local copy, the only version of a web service

is the one which is available online. If a change occurs in a static API, this will not affect the local copy and it will not disrupt the client application's functionality. However, if a web service is changed in an incompatible way, then this constitutes a breach in the communication contract (*i.e.,* the interface) between the service and the client, even if the medium of the communication (*i.e.,* the Internet) remains intact and this may affect the client application's functionality.

Economics: Given that web services are considered self-contained business applications, their respective owners have financial and economic motives when they develop and maintain their software. The independence of the service components also implies that their owners are self-interested and self-motivated. Any decision made around the evolution of a service or the adaptation of a client application will have to be based both on technical and socio-economic criteria. For example, if a provider decides on a change that may potentially lead to excessive adaptation costs for the clients, then the latter may consider abandoning the provider for a competitor. In reality, a service-oriented system operates within a broader business ecosystem, where multiple providers offer similar services and clients will have to choose one of them. Keeping up with competition and maintaining backwards compatibility are two conflicting goals, where achieving the former includes the risk of losing clients, while achieving the latter includes the risk of technological stagnation and staying behind with respect to competition. It is essential that the trade-off between these to goals is considered when making evolution decisions.

Although service components are independent in principal, there are very strong ties between them, both technical and economical. Software-engineering best practices, in this context the preservation of backwards compatibility, cannot or may not always be followed. Even if we assume that best practices can be followed, this is not always the case and a breaking change can have negative technical and financial repercussions for the clients. Furthermore, the imprecise definition of certain service-oriented styles may be interpreted differently by the providers resulting in variability of syntaxes and service interfaces. In this case, the design of a service-oriented system may become more

rigid. Clients may get locked with a particular provider without the flexibility to satisfy their requirements with a different service. Therefore, there is a need to support the clients with automated tools to react to service changes in as effective and expeditious manner as possible.

Research Problem 1 (RP1): How can clients be supported to adapt to change or to a new service effectively and efficiently?

As web services are implemented using a variety of technologies and architectural styles, like WS-* services or REST services, different tools and specifications have been developed for these styles. Although the problems around service evolution and its impact on client applications are similar across the different styles, the variety of tools gives different capabilities to create solutions for adapting client applications against changed web services. Therefore, along with developing generic methods to support service evolution, there is also a need to create and implement specific tools to facilitate the adaptation of client applications.

Research Problem 1-WS-* (RP1-WS): How can client applications be supported in adapting to changed WS-* services?

Research Problem 1-REST (RP1-REST): How can client applications be supported in adapting to changed REST services?

Additionally, the technical repercussions to clients may lead to respective financial repercussions to the providers. Therefore, another need is to support technical and business decisions on an evolving service ecosystem. Such a support system will be required to take a holistic consideration of the ecosystem, its needs and predict its actions and its reactions concerning evolution.

Research Problem 2 (RP2): How can providers and clients be supported in making decisions on service evolution with both technical and economic criteria?

1.3 Technical Background

In this section, I describe certain technologies and tools that are going to be referenced throughout this document and are necessary for the understanding of my work.

*WS-** (W3C, 2013) (Web of Services) is a message-based design proposed by W3C to construct modular and service-oriented systems based on a set of technologies and standards. WS-* services use SOAP (Simple Object Access Protocol) (W3C, 2007a) as the message exchange protocol for structured information. SOAP is characterized (a) by *extensibility* (W3C, 2007c), so that other specifications can define additional features for the protocol like "security", "routing" and "correlation", (b) by *neutrality* (W3C, 2007b), since it can work with a variety of communication protocols on the application layer, not limited to HTTP and (c) by *independence* (W3C, 2007c), as it allows for any programming model. SOAP uses exclusively XML to encode its messages in which it specifies the remote calls and response information. SOAP was criticised for verbosity and that was one of the reasons, why REST gained popularity over WS-* services. The other reason is the programmatic simplicity and ease of use of REST similar to making HTTP calls. WS-* services are generally perceived as operation-centric, where any kind of operation can be invoked remotely using the appropriate input and the output is not required to be persistently stored.

REST (Fielding, 2000) (Representational State Transfer) is an architectural style built on top of the HTTP communication protocol. REST imposes certain architectural constraints that effectively remove responsibilities from the server, other than delivering the requested functionality. In practice, the requests carry no information about the state of the client and the client may cache the requests. In turn, the hyperlinks determine the state of the service. REST is implemented exclusively over HTTP, while the most popular format for information exchange is JSON, although XML may also be used. REST services are data-centric, where everything is perceived as a resource, upon which a client can perform CRUD operations (**C**reate, **R**etrieve, **U**pdate

and **D**elete), which conceptually correspond to the HTTP methods (`PUT`, `GET`, `POST`, `DELETE`). The response of these operations is usually a view of some persistently stored data.

WSDL (W3C, 2001) (Web Service Description Language) is an XML-based language to define interfaces especially for WS-* services, although WSDL2.0 can also specify interfaces for REST services. The WSDL defines a service as a set of operations with input and output parameters. The WSDL includes or may reference an XML schema that describes the types and the structure of the operation parameters. The concrete part of the WSDL interface defines the service endpoint and the communication protocol used for each operation.

WADL (Hadley, 2009) (Web Application Description Language) is an XML-based language to define web applications that are built based on the REST architectural style. According to WADL, a REST service is a collection of resources upon which only HTTP methods can be performed. For each method, the interface specifies the request and the response parameters and, similar to WSDL, a XML schema defines the parameters' types and structure.

Apache Axis2 (Apache, 2012) is an engine especially designed for WS-* services and Java client applications. Axis2 offers the capabilities of generating a WSDL interface from Java source code and a Java client proxy from a WSDL interface. In the former case, the user has to specify what methods are going to be exposed as service operations from a single class and the parameters have to be of concrete types, because the XML schema cannot be inferred for abstract classes or interfaces. In the latter case, Axis2 generates a single class that contains all the operations of the WSDL interface and all the parameter types are implemented as inner static classes within the proxy. The client application can then construct the input data according to the generated classes and invoke the service operations from the proxy.

WADL2Java (Oracle, 2013) is a tool offered as part of the Jersey framework for REST services by Oracle, similar to Axis2 for WS-* services. In order to generate a WADL interface, the tool first adds special annotations to the methods and the class that are to be exposed. The generated proxy is a Java class that follows the structure of the resources defined in the interface

and each resource includes the available methods. However, the various types included in the XML schema are generated as separate classes.

The challenges around service evolution and the methods proposed to address them in this thesis apply to service-oriented systems regardless of the implementation style. However, the WSDarwin tools are implemented specifically for each style (REST and WS-*) considering their peculiarities and taking advantage of the artifacts and technologies for each one of them (e.g. WSDL and Axis2 for WS-* and WADL and WADL2Java for REST). Pautasso et al. (2008) give a comprehensive comparison between WS-* and REST, which actually emphasizes the point that different styles require different tooling support.

1.4 Thesis and Contributions

The thesis of this work is that *"service system evolution can and should be, not only technically, but also socially and economically conscious through support from automated software engineering tools."* Today, when deciding on, developing and implementing a change not all of these considerations are taken into account at the same time nor with the same priority. Automated tools to support the evolution and adaptation of service components can help to address this inadequacy.

In my work, I have developed *WSDarwin* as a means to implement my thesis and prove its plausibility. Towards this goal, WSDarwin makes two technical contributions to address the first research problem for WS-* (**RP1-WS**) and REST (**RP1-REST)** services and support the adaptation of client applications, and a theoretical contribution to address the second research problem (**RP2**) and support the decision-making process of service-ecosystem participants on service evolution.

For the first technical contribution and towards addressing the first research problem for WS-* service clients (**RP1-WS**), WSDarwin offers an *integrated* set of two tools providing a *complete* solution for *systematic* adaptation of WS-* client applications, in the event of service evolution.

- The **service-interface comparator** automatically identifies the differences between two versions of a web service. The comparator uses a proprietary representation for the service interfaces, which is less verbose than WSDL or WADL, in order to be faster and to be able to handle both WS-* and REST services seamlessly. The representation is based on the **WSMeta** web service interface meta-model, which I also propose in this thesis. The comparison algorithm can identify simple CRUD (Create, Retrieve, Update, Delete) changes as well as more complicated ones like refactorings.

- The **client-application adaptor** automatically adapts client proxies to new versions of a service interface. The tool receives the differences from the comparator as an input and uses them as an edit script to adapt the old client proxy to the new version of the service. The adaptation algorithm is generic enough to handle any kind of change and regardless of the client's underlying programming language. Not all service clients may use a client proxy to invoke a service, but, especially for WS-* services, this is the easier and more common way. Therefore, the adaptor can support a large number of service clients.

The **WSMeta** service specification is proposed to provide a more abstract and less verbose representation for service interfaces than WSDL or WADL specifications. This leads to more efficient comparison methods. At the same time, this specification follows the structure of the generated client proxy, thus, the service interface annotated with the results of the comparison can be seamlessly consumed by the adaptation process, which by extension becomes more efficient and effective in addressing changes from the client's side. Additionally, the specification has the ability to represent services of the two popular styles that this thesis deals with; WSDL and WADL service interfaces. To achieve this, we used VTracker (Fokaefs et al., 2011), a generic XML differencing method, on the respective schemas of the two specifications to identify the best mapping between their elements. Merging the two schemas according to the VTracker results gave the WSMeta specification.

The WS-* support of WSDarwin is implemented as an **Eclipse plug-in**. The plug-in offers a complete solution to the problem of service evolution guiding the developers of client application from understanding how the service changed to automatically adapting the client application and testing it to confirm that the changes were benign. The user's interaction with the plug-in is very simple (one button per action) and all the actions are orchestrated so that very little input is required from the developer.

For the second technical contribution and towards addressing the first research problem for REST service clients (**RP1-REST**), WSDarwin offers a set of four tools providing support for the development and maintenance of REST applications and, among others, address the absence of formal specifications.

- The **service-interface specification generator** automatically exercises a web service and analyses the requests and responses to infer the schema and the structure of the service. This tool is specifically targeted to REST services, for which a service interface is not always provided in a standard format.

- The **service-interface comparator** is the same as the comparator for WSDL interfaces. This is due to WSMeta which has the ability to handle handle both WSDL interfaces for WS-* services and WADL interfaces for REST services seamlessly.

- The **client-proxy generator** automatically generates service client proxies from service interfaces. WSDarwin integrates Oracle Jersey to generate proxies from WADL service interfaces. This integration allows the developers to invoke said tools from within their development environment and their output can be then used by the other WSDarwin tools with minimal human input.

- The **cross-vendor service mapper** semi-automatically maps the responses of two services by different providers within the same domain. The tool assumes that, since the services belong to the same domain,

some of response elements will have the same or similar values and therefore will correspond to the same entity. The goal of the tool is to map these elements and indicate to the developer how the client application should be changed to migrate from one service to another.

The REST support of WSDarwin is implemented as a stand-alone web application. The application is developed in the spirit of REST to be easy to use and requiring little and readily available input. The tools are complemented with a set of visualizations to allow the users to change the generated interface at will and better identify the differences between services.

The theoretical contribution of my research is setting the foundations for a framework to support the decisions of service providers and clients around the evolution of web services and the adaptation of client applications. The argument of this research is that while evolution decisions may be motivated by technical drivers, such as new emerging technologies and changing requirements, they are actually dictated by economic and financial parameters. Furthermore, as modular and distributed systems, service-oriented systems operate within a business ecosystem and every decision may have an impact on part or on the whole ecosystem. Therefore, it is necessary for the decision makers to adopt a holistic approach and take into account both the technical and the economic aspects of service evolution. Towards supporting the decision-making process on service evolution, my thesis makes three distinct contributions.

- It proposes an **economic model** to calculate and estimate a number of non-technical parameters relative to web services. The model consists of functions and a series of optimizations to calculate the costs of evolution and adaptation, optimal evolution effort and the price and the value of the new version of the service for every possible evolution scenario. The model takes into account the actions and reactions of all the participants of the service ecosystems, clients and providers alike, and the indirect effect of each decision to the rest of the ecosystem.

- It proposes a **game-theoretic model** to capture all the interactions and conflicting interests of the service ecosystem participants when they decide about their evolution strategies. Their interactions are modelled as a two-stage game, where the providers decide first on their evolution strategy and the clients react by choosing their preferred provider.

- It proposes a **decision-support system**, which simply solves the aforementioned game and finds its Nash equilibrium. This equilibrium determines the best action for each participants as the response to other participants' actions and the state of the ecosystem. It also dictates the evolution strategy of each provider, the division of the market (what client uses which provider), evolution and adaptation efforts for changed services and the prices for the new versions.

The use of the models and the decision-support system is demonstrated with a synthetic but realistic case study from the domain of cloud services inspired by real cloud providers like Amazon, Google and Microsoft. In this example, we explore a variety of evolution scenarios and their impact on the ecosystem. The primary goal of this part of my thesis is to raise awareness about the importance of economics and business relationships on the strategic decisions concerning the lifecycle of software and especially of web service systems.

1.5 Outline

This dissertation is organized in 4 chapters; three on the contributions of WSDarwin and one that concludes the work. Each contribution chapter opens with an introduction to summarize the research problem and to lay out the papers that I have published or submitted around this problem. The purpose of the introduction is also to connect the papers with each other and describe the contributions that each one of them makes towards addressing the single research problem. The rest of each chapter contains the papers as they have been published or submitted.

The dissertation is organized as follows. Chapter 2 describes the first technical contribution of this thesis on **RP1-WS** to support clients of WS-* services. More specifically, the chapter presents an empirical study on the evolution of several industrial WS-* services (Section 2.1) to examine how services generally evolve and the impact of these changes to service clients. Next, the chapter describes a methodology to automatically identify changes between two versions of a service by comparing the public WSDL interfaces of the service (Section 2.3). The chapter also presents an algorithm and a tool to automatically adapt client applications to new versions of a service using the previously identified changes between the two versions (Section 2.4). Finally, the chapter demonstrates an Eclipse plug-in that implements a complete solution to support WS-* clients from comparing service interfaces to adapting and testing Java clients (Section 2.5). The performance of the tool and its different features was evaluated on the Amazon EC2 SOAP service.

Chapter 3 describes the second technical contribution of the thesis on **RP1-REST** to support clients of REST services. First, the chapter examines how REST services are developed and published to clients using the Tumblr API as a case study (Section 3.1). In the context of this study, the section also presents the WSDarwin method to automatically generate the WADL interface for a REST service, which can be used for several tasks, including the generation of client proxies and the comparison of service interfaces. Second, the chapter describes a methodology to automatically map similar services from the same domain but offered by different providers based on the similarity of their input and output data in order to support the migration of client applications to services of different providers (Section 3.2). Finally, the chapter demonstrates the implementation of the interface generation, the comparison of versions of a service and the cross-vendor comparison as a simple and easy-to-use web application (Section 3.3).

Chapter 4 describes the theoretical contribution of the thesis on **RP2** to support the decision-making process around the evolution of service systems taking into account both the technical and economic aspects of the problem, the complex interactions between providers and clients and their conflicting

interests. The chapter studies the economic parameters of software evolution and the interactions between one provider and one client in various software ecosystems and models these interactions as a game (Section 4.1). Based on this analysis, the chapter describes a simple tool based on decision trees to guide the provider to make the optimal decision that will maximize both the provider's and the client's payoff (Section 4.2). Finally, service evolution was studied in a more complex ecosystem with more providers and clients, where again the interactions of the participants are modelled as a game and an economic model is proposed to calculate costs, values and prices of services (Section 4.3). The use of the decision-support system is demonstrated in a realistically synthetic cloud ecosystem.

Finally, the dissertation is concluded in Chapter 5, where a summary of the contributions and the impact of the thesis is provided. The chapter also outlines how the work in this thesis can be extended by future research.

Chapter 2

Support for WS-* Service Clients

WS-* services are generally characterized by their degree of structure, especially compared to their REST counterparts. The nature of the message exchange protocol and the complexity involved in constructing SOAP messages necessitate the exposure of the service through a structured and machine-readable interface and the use of a dedicated middleware to consume this interface. This also dictates the use of good practices in designing service-oriented systems and increases the degree of automation in the development and maintenance of service applications. The ability for increased automation has led to the development of a number of tools to support the development of web services and client applications. These tools support the automatic generation of the WSDL interfaces from source code, the generation of client code from the interface, testing environments to invoke the service, graphical editors to design and develop service applications and so on.

Concerning the evolution of service-oriented systems, the structured nature of WS-* services can help developers of client applications to better reason about the changes to a service and address these changes more effectively. Moreover, the ability to automate certain tasks will ensure the efficiency with which changes have to be addressed, since service systems are online systems and continuous communication between services and clients needs to be maintained. In my work, I have developed methods and tools to address research problem 1 for WS-* services (**RP1-WS**). The WSDarwin toolkit is imple-

mented to support the automatic adaptation of client applications against evolved WS-* services. The tools take advantage of existing standards and other automated tools (like WSDL and Axis2). WSDarwin offers a complete solution from identifying the changes in the service to adapting and testing client applications. The tools are integrated under the same interface, fully automated and yet interactive in order to minimize the necessary effort by the developers. At the same time, the tools guarantee that the applications will be correctly adapted to the new version and they will be fully functional.

The first step towards supporting client applications during the evolution of web services is to understand how WS-* services evolve and what is the impact of these changes to client applications that consume the services. To this end, I conducted an empirical study (Fokaefs et al., 2011) (Section 2.1), in which I examined different versions for a number of WS-* services from various providers, including Amazon, FedEx, Bing and Paypal. First, I correlated the changes in the services with relevant business announcements from the providers and the conclusion was that most changes are introduced to add new functionality or extend existing functions with extra data. Second, I studied how the services changed and whether their elements were added, deleted or changed. The study showed that there are two extremes with respect to the evolution of a web services. On one hand, we have a more conservative evolution strategy, where the provider aims for changes that have the smallest possible impact on the interface of the service, thus maintaining the backwards compatibility between subsequent versions. On the other hand, we have providers that aim to maintain a concise and well-designed web service, in many occasions, at the expense of client applications. Therefore, there is a tradeoff in the decision towards choosing one of the two directions. Among the examined providers, there are two representatives for these two extremes; Amazon for the former strategy and FedEx for the latter. Amazon's changes have a small effect on the interface of the service, *i.e.,* its contract with its clients, and most of the changes concern the addition of new functions that don't affect existing functionality. According to Xing and Stroulia (2005a), the Amazon web service is in a growing state in its evolutionary history. Although this is partly sup-

ported by the business decisions by which some of the changes are motivated, the general history of the service tells us that this growing trend is mainly to maintain the backwards compatibility and not always to add new functionality to the service. On the other hand, FedEx seems to frequently breach the contract between the clients by deleting, renaming and generally changing a large number of each elements. This results in shorter interfaces (compared to Amazon, for example) and a more concise design, but with a higher risk to clients. Finally, the study of the evolution of these services revealed a set of recurring evolution scenarios, which were described along with their potential impact on client applications.

The empirical study revealed three needs towards automating the support for client applications.

1. The WSDL interface may be too verbose and only a portion of the specified information is actually needed to identify the changes between two versions.

2. There is a need for an efficient and domain-specific method to compare two versions of a service's interface.

3. There is a need to correlate the differences from the service interface to how the clients consume the interface (through an auto-generated client proxy) and then automatically adapt the client application to the new version.

To address the first need, I developed the **WSMeta** meta-model (Fokaefs and Stroulia, 2013b) (Section 2.2) to express the interface of a web service in a more concise manner and specifically address the task of identifying differences between two versions. WSMeta ignores the syntactic elements of the interface and includes only the elements that correspond to the service's functionality. A service interface in WSMeta is expressed as a collection of operations with input and output data. Therefore, this representation focuses only on the elements that are important for the evolution of the service. Additionally, it replaces references with containment by copying the referenced elements inside

the elements that refer to them. These two properties improve the efficiency with which the interfaces are translated into WSMeta and with which the WSMeta representations are read and compared with each other. WSMeta was developed to have the ability to express both WSDL interfaces for WS-* services and WADL interfaces for REST services.

Having a compact representation for service interfaces, I continued to develop the **service-interface comparator** (Fokaefs and Stroulia, 2014b) (Section 2.3) to identify the differences between two versions. The WSDarwin comparison method includes formal rules to identify basic differences, *i.e.,* additions, deletions and changes, but also more complicated, refactoring-like, differences, such as renamings and moves. The method is also based on certain heuristics in order to improve its accuracy and efficiency. For example, if two elements have the same identifier between the two versions, they are considered to be one and the same element. In fact, it was shown that WSMeta, the heuristics and the domain-specific nature of WSDarwin greatly improve its efficiency compared to domain-agnostic methods. I compared WSDarwin with VTracker (Fokaefs et al., 2011), a domain-agnostic method to compare tree-like structures (such as XML or WSDL files). The results showed that, first, WSDarwin was much faster than VTracker and that its execution time is linear to the size of the interfaces, compared to the exponential execution time of VTracker.

Having identified the differences on the interface of the service between two versions, I developed an algorithm, the **client-application adaptor** that is able to automatically adapt the client application based on the changes (Fokaefs and Stroulia, 2012) (Section 2.4). The algorithm is designed so that it is independent of the change and the programming language in which the client is developed or the particular tool that was used to generate the client proxy. The philosophy of the algorithm is to alter the client proxy corresponding to the old version of the service interface in order to invoke the proxy of the new version of the service interface. In practice, this means that the actual client application will be invoking the service through the old interface but accessing the new functionality. Since no changes are made to the actual client code, but

only to auto-generated code, the adaptation process is non-invasive and it does not affect the developer's awareness of their own code. The goal is to make the client invoke the new version by sending the input data that is already available and still valid and by processing the output data that was needed in the previous version as well. Therefore, the algorithm copies the data of all the old input variables to the new input variables, uses those to invoke the new version of the service through the proxy and then repeats the process for the output variables as well. Newly added variables are assigned default values and deleted elements are ignored. If the new variables are optional, then the defaults will have no impact in the invocation of the service. Otherwise, if the default values are semantically meaningful, this will be confirmed by executing the client's test cases.

The automatic adaptation process is not expected to create design overhead, since adapters are created between the current version of the service that the client uses and the most recent version. If a newer version is published, the process will not be applied on the previous version, but again on the current version, with respect to the client. This implies that the client will only have to maintain two versions of the service. However, this process may have an overhead to the client's understanding of the web service. In this case, the user may opt to manually adapt to the most recent version and completely replace the current version.

As it was discussed in the empirical study (Section 2.1), there are three broad categories of changes according to their potential impact to client applications. First, there are changes that have *no effect* to either the interface or the functionality of the web service. These can be refactorings that do not affect the signature of the public operations or the structure of the input and output types. The second category includes *adaptable* changes that affect the structure of the service interface but not its functionality. These changes can also be refactorings, but which may affect the public interface of the service. Finally, we can have changes that are *non-adaptable* and require additional effort from the developer of a client application. These changes usually affect the functionality but not necessarily its public interface. In the case, when

the interface is not affected, the change is harder to identify because it doesn't throw any exceptions but its output does not correspond to that expected by the client. Therefore, the developer will have to use the client's test cases to confirm what causes the application to fail and why. The adaptor proposed in this thesis can address any change that affects the interface of the service. In case of adaptable changes, the process ends there and the adaptor is enough to adapt client applications. In case of non-adaptable changes, the user needs additionally invoke the test cases. Combining the results of the comparison and the test cases, the user will be able to understand what and how was changed and where and why it caused the client application to break, thus enabling him to efficiently and effectively address the adaptation manually.

The WSDarwin tools to support the evolution of WS-* service clients are implemented as an **Eclipse plug-in** (Fokaefs and Stroulia, 2014a) (Section 2.5). The plug-in follows the "one-click" approach; each function (comparison and adaptation) is implemented as one button. This minimizes the effort required by the developer and facilitates the whole adaptation process. The developer would need to already possess the two versions of the service interface, which are provided as input to the tool. The diff script produced by the comparison tool is then fed to the adaptation tool. The adaptation process is applied on Java client proxies generated by the Apache Axis2 tool. A parsing tool is used to map the WSMeta elements from the diff script to the proxy elements in order to alter them accordingly for the adaptation. The plug-in offers a third button to invoke the JUnit test cases of the client, assuming that these are already provided, in order to confirm that neither the adaptation process nor the evolution of the service affected the client application. The plug-in was evaluated on the Amazon EC2 web service and it was shown that the tool maintains a short execution time even for relatively large service interfaces. The most expensive task is the adaptation, mainly due to the complexity of resolving references between the diff script and the client proxies and refactoring the client proxy. The second most expensive task is parsing the raw WSDL interfaces to WSMeta representations. Naturally, this task depends (linearly) to the size of the interfaces. The execution time for

comparing the service interfaces is negligible.

The methods proposed and developed in this thesis for WS-* services, the comparator, the adaptor and WSMeta, are generic enough and are based on assumptions that hold for most technologies. For example, the assumption that service interface can be perceived as collections of operations with input and output (as represented in WSMeta) to follow the client proxy structure is true for proxies produced for any language and not only for Java proxies. Naturally, the implementation of the methods in tools for specific languages depending on the provided tools and infrastructure for these languages, including proxy generators and code manipulators (like JDK for Java).

2.1 An empirical study on web service evolution

Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A., 2011. An Empirical Study on Web Service Evolution. In: IEEE International Conference on Web Services (ICWS 2011). ICWS '11. Washington, DC, USA, pp. 49-56.

An Empirical Study on Web Service Evolution

Marios Fokaefs, Rimon Mikhaiel, Nikolaos Tsantalis, Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
{*fokaefs,rimon,tsantalis,stroulia*}*@ualberta.ca*

Alex Lau
Center for Advanced Studies
IBM Toronto Lab
Markham, ON, Canada
alexlau@ca.ibm.com

***Abstract*—The service-oriented architecture paradigm prescribes the development of systems through the composition of services, i.e., network-accessible components that are completely specified by (and invoked through) their WSDL interface descriptions. Systems thus developed need to be aware of changes in, and evolve with, their constituent services. Therefore, accurate recognition of changes in the WSDL specification of a service is an essential functionality in the context of the software lifecycle of service-oriented systems.**

In this work, we present the results of an empirical study on WSDL evolution analysis. In the first part, we empirically study whether VTracker, our algorithm for XML differencing, can precisely recognize changes in WSDL documents by applying it to the task of comparing 18 versions of the Amazon EC2 web service. Second, we study the changes that occurred between the versions of various web services and discuss their potential effect on the maintainability of a general service system.

***Keywords*-component; formatting; style; styling;**

I. Introduction

Service-system evolution and maintenance is an interesting variant of the general software-evolution problem. On one hand, the problem is quite complex and challenging due to the fundamentally distributed nature of service-oriented systems, whose constituent parts may reside not only on different servers but also across organizations and beyond the domain of any individual entity's control. On the other hand, since the design of a service-oriented system is expressed in terms of the interface specifications of the underlying services, the only changes that the overall system needs to be aware of are those that impact these interface specifications; any changes to the service implementations that do not impact their interfaces are completely transparent to the overall system. In effect, the WSDL specifications of the system's constituent services serve as a boundary layer, which precludes service-implementation changes from impacting the overall system. Further simplifying the problem is the fact that service providers, once they have published and made available to clients their service-interface specifications, are highly motivated to refrain from making changes that may threaten the stability of their clients, especially since, frequently, they do not necessarily know by whom, how often or by how many clients they are used.

Therefore, changing a service is a rather sensitive task. Although changes might still need to happen and the system to evolve, the service provider needs to be aware of the impact a specific change might have on existing clients. The impact is with respect to factors such as time and human effort to deal with the change as well as business decisions (e.g., a client should not break if it belongs to a strong partner). If the impact is low, the service provider might still need to provide some backward compatibility. If the impact is high but the change is still necessary, then its effect on a client might need to be leveraged by the client's developer with the use of appropriate tools and techniques.

In this scenario, the objective of precisely recognizing the changes to the WSDL specification of a service interface and their impact on client applications is highly desirable and precedes the process of actually dealing with the change either on the server or on the client side. Further, assuming that such a precise method for service-specification changes existed, it would be extremely useful if one could (a) characterize the changes in terms of their complexity and (b) semi-automatically develop adapters for migrating clients from older interface versions to newer ones.

In our work, we have developed VTracker, a tree-alignment algorithm. VTracker is an evolution of SPRC [1], an algorithm developed for the task of RNA secondary structure alignment. VTracker (and SPRC) are based on the Zhang-Shasha's tree-edit distance [2] algorithm, which calculates the minimum edit distance between two trees given a cost function for different edit operations (e.g. change, deletion, and insertion). In our earlier work [3], we have already applied VTracker to the task of comparing web-service specifications. However, in this earlier work, our objective was to illustrate how VTracker could be used to compare BPEL specifications. In the mean time, we have evolved VTracker to enable it to compare large XML documents, which has allowed us to use it in comparing complex WSDL specifications in the empirical study reported in this paper.

More specifically, in this paper, we are interested in analyzing the long-term evolution of real world services, including the Amazon Elastic Cloud Computing (Amazon

EC2)[1], the FedEx Package Movement Information and Rate Services[2], the PayPal SOAP API[3] and the Bing search service[4]. First, we have applied VTracker to the problem of pair-wise comparison of subsequent versions of these service-interface specifications. We manually inspected the results of the comparison in order to assess how effective VTracker is for the purpose of accurately recognizing service-interface changes. Next, we examined the various types of changes that the algorithm identified in the history of the real-world services we study, in order to understand how services evolve, what types of changes are more or less frequent, and whether these changes endanger the stability of the clients.

The rest of the paper is organized as follows. In Section II we give an overview of VTracker and we elaborate on how this tree-differencing algorithm works. In Section III, we discuss our mapping of WSDL documents to tree representations that can be understood and compared by VTracker. In Section IV, we evaluate VTracker's ability on studying service evolution. In Section V, we discuss the results of our study and we present some interesting change scenarios. In Section VI we review the related literature and finally in Section VII we conclude our work and discuss a few of our future plans.

II. VTracker Outline

VTracker is an extension to Zhang-Shasha tree-edit distance algorithm. Zhang-Shasha's tree edit-distance algorithm [2] given a cost for *change*, *insertion*, and *deletion* operations, computes the cheapest cost to transform one tree to the other with an average complexity $|T_1|^{3/2} \cdot |T_2|^{3/2}$, where $|T_1|$ and $|T_2|$ are the sizes of the two trees respectively. VTracker uses this algorithm as a starting point, and it extends it in two ways. First, VTracker reports the least expensive edit script that transforms one tree to the other. Second, it allows for move operations, through a post-processing phase of mapping deleted sub-trees from the first tree to inserted subtrees of the other.

The result of a comparison between two trees – T_1 and T_2 – is a *tree-edit script*, i.e., a sequence M of mappings, $map(i, j)$, where i is a node in T_1 and j is a node in T_2, such that $\forall (i_1, j_1) and (i_2, j_2) \in M$:

- $i_1 = i_2$ iff $j_1 = j_2$; each node cannot be involved in more than one edit operation;
- $T_1[i_1]$ is on the left of $T_1[i_2]$ iff $T_2[j1]$ is on the left of $T_2[j_2]$; the mapping preserves the original sibling order;
- $T_1[i_1]$ is an ancestor of $T_1[i_2]$ iff $T_2[j_1]$ is an ancestor of $T_2[j_2]$;it also preserves the ancestor-child order.

[1]http://aws.amazon.com/ec2/

[2]http://www.fedex.com/us/developer

[3]https://www.paypalobjects.com/en_US/ebook/PP_APIReference/architecture.html

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

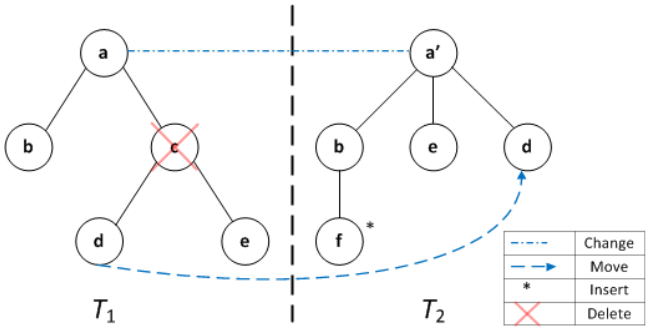


Figure 1. Tree Edit Script.

Let us illustrate the properties of the labeled-ordered tree-differencing algorithm with the example shown Figure 1. The difference of trees T_1 and T_2 shown in Figure 1 is the edit sequence $M = (b, b), (d, d), (e, e), (c, -), (-, f), (a, a')$. This sequence consists of the following edits: node a is changed to a', node c is deleted, node f is inserted, and node d is moved. Both nodes b and e are unchanged at the second tree. This solution obeys the above constraints as it maps (a, a') where both a and a' are ancestors of all other nodes; additionally, (b, b) and (e, e) preserve the sibling order where in both trees, node b is on the left of node e. Clearly, moved nodes do not preserve the original ordering relations.

A. Affine Cost

The original Zhang-Shasha algorithm assumes that the cost of any deletion/insertion operation is independent of the operation's context. Thus, the cost of a node insertion/deletion is the same, irrespective of whether or not that node's children are also deleted/inserted. As a result, it will consider as equally expensive two different scripts with the same number and types of edits, with no preference to the script that may include all the changes within the same locality. Such behavior is unintuitive: a set of changes within the same sub-tree is more likely than the same set of changes dispersed across the whole tree.

In order to produce more intuitive tree-edit sequences, we have modified the Zhang-Shasha algorithm to use an *affine-cost policy*. In VTracker, a node's deletion/insertion cost is context sensitive: if all of a node's children are also candidates for deletion, this node is more likely to be deleted as well, and then the deletion cost of that node should be less than the regular deletion cost. The same is true for the insertion cost. To reflect this heuristic, the cost of the deletion/insertion of such a node is discounted by 50%.

B. Simplicity Heuristics

It is very likely to have many edit scripts associated with the calculated edit distance. Thus, the objective of VTracker simplicity filter is to discard the unlikely solutions from

the solution set produced by the VTracker tree-edit distance algorithm through a set of simplicity heuristics.

The first simplicity heuristic advises the algorithm to "prefer minimal paths": when there is more than one different path with the same minimum cost, the one with the least number of deletion and/or insertion operations is preferable.

The second simplicity heuristic advises the algorithm to "prefer contiguous similar edit operations". Intuitively, this rule says that contiguous same-type operations could be considered as a single edit operation. When there are multiple different paths with the same minimum cost and the same number of editing operations, the one with the least number of changes (refractions) of operation types along a tree branch is preferable.

The third simplicity heuristic advises the algorithm to maximize the number of nodes along a tree branch to which the same edit operation is applied. VTracker proposes that, to the extent possible, sibling nodes should also suffer the same edit operations.

III. Applying VTracker to WSDL Documents

The WSDL specification is quite verbose. Consider, for example, the mapping of a single public class method (implemented in Java) into a WSDL operation. This mapping will produce a tree rooted at the operation element which will contain a number of messages corresponding to the number of parameters in the method signature. Each of these message elements, in turn, will contain a single part element, which in turn will refer to a data type. In fact, there are several cases within a WSDL document where an element contains a single element; clearly all these cases cause the implicit tree representation to become deeper without necessarily adding any information content to it. Such deeply nested trees can, in fact, severely compromise the performance of VTracker.

This is why for the purpose of comparing WSDL specifications with VTracker, we developed an intermediate XML representation, much simpler than WSDL which still captures the information content relevant to our task. This simpler representation includes information about data types and their use in operations. This is because we are interesting in study the web service evolution from the client's perspective and identify what changes are easily adapted and which are not. Operations and types are the interesting parts since the operations are the only point of interaction between the client and the web service and the types are directly related to the operations (through input and output).

Thus, given a WSDL document, in order to construct its simpler XML representation, we perform the following steps to it:

1) We strip the files off their functional parts such as the SOAP bindings.
2) We trace the references from the operations' inputs and outputs to the types through the messages and the xs:elements. We replace the messages in the inputs and outputs with the corresponding types, thus eliminating messages and xs:elements.
3) We remove the messages as they essentially are mediators from types to operations and they add no additional information.
4) We remove the xs:element nodes which are immediate children of the root of the file. This is because these nodes serve as mediators between the types and the messages.
5) Finally, we remove any annotations or documentation nodes in the file. This data is irrelevant to the purpose of this study.

To perform these changes we used XSL transformations. Generally, WSDL files from different services have slightly different structures thus special XSLTs had to be created for each service. For step 2 of the transformations, we took advantage of the naming conventions in the files in order to make the XSLT as simple as possible. For example, in many cases the name of the type that is used by a message is usually the word "Type" appended to the name of the message.

Eventually, we got valid XML files (although not valid WSDL files), which are now closer to WADL-like files. The goal of performing these changes was to to minimize the number of nodes and eliminate as many levels of indirection as possible. This will relieve us from any unnecessary data, such as messages and bindings, and help VTracker produce more concise results faster.

IV. Evaluation of VTracker

The first question we want to answer in this work is to examine whether VTracker can be used to accurately recognize the evolution of web services. To that end, we employed a clustering technique. We compared the 18 versions of the Amazon EC2 service pairwise, for example, version 1 against version 2, version 2 against version 3 and so on. VTracker then produced the tree edit distances between every pair of operations for the given pair of versions. We applied a hierarchical agglomerative clustering algorithm. In order to run the clustering between the two versions, we need to have distances between all operations (from both versions) in a square distance matrix. Thus, we have to construct the total distance matrix as shown in Figure 2. Although the individual sub-matrices might not be square, for example, because new operations were added from one version to the next, the concatenation of the matrices produces a square distance matrix which can be used by the clustering algorithm. We used the R Project for Statistical Computing[5] to run our clustering analysis.

Using such a technique, we anticipate that different versions of the same operation will be grouped in the same

[5]http://www.r-project.org/

cluster as being similar to each other, a fact that will be recognized by the small distance that VTracker will calculate between them. In case this does not happen, i.e., if two versions of an operation are clustered in different clusters, we will assess whether the difference between two versions is so significant that the two operations cannot be considered as versions of the same operation but rather two distinct operations; one that was removed from the first file and another new one that was added in the second file. Closer examination of the cluster will help us better understand the changes between versions and the motivation behind them and may also reveal the need to tune VTracker with domain specific knowledge.

We compared the 18 versions of the Amazon EC2 service pairwise, for example, version 1 against version 2, version 2 against version 3 and so on. VTracker then produced the distances between every pair of operations for the given pair of versions. We applied a hierarchical agglomerative clustering algorithm. In order to run the clustering between the two versions we have to construct the total distance matrix as shown in Figure 2. Although the individual sub-matrices might not be square, for example, because new operations were added from one version to the next, the concatenation of the matrices produces a square distance matrix which can be used by the clustering algorithm. We used the R Project for Statistical Computing[6] to run our clustering analysis.

Figure 2. The construction of the total distance matrix.

The results of the clustering experiment were in forms of dendrograms (Figure 3) so that we can see how the operations of two subsequent versions were grouped together. In Figure 3, we see a particular dendrogram for the comparison between the 4th and the 5th version of the Amazon EC2 web service. The operations are indexed 1-19 for version 4 and 20-39 for version 5. As it becomes obvious operation 1 in version 4 corresponds with operation 20 and so on. The height of the tree corresponds to the level of the distance where the clusters merged. We can distinguish three cases of clusters:

- The operation with index 39 does not belong to a cluster until a very high distance. This is because this particular operation was a new addition in version 5 and therefore there was not an available good mapping.
- There are operations that are paired with each other in a distance higher than 0. This is because these operations or the types that they are using changed from one version to the next. VTracker was still able to map them correctly and report their changes through their distances. As it can be noticed the pair 19-38 stands even higher than the rest of the modified operations. This is because in this particular case the type that was immediately used by the operation changed (*shallow change*), while in the rest of the cases the changes occurred deeper in the chain of types and thus the effect of the change was "diluted" along the various elements (*deep change*).
- The rest of the operations which are paired in distance 0. These are operations that remained the same between the two versions and their types did not change either.

This experiment helped us confirm that VTracker is able to correctly map elements between different versions of the same service and identify possible changes between them.

V. Study of Web Service Evolution

In this part of the study, we used VTracker as tool to report all changes that happened between different versions from a set of services. For our study, we chose to examine the evolution of the following services:

- **Amazon EC2**. The Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. We studied the history of the web service across 18 versions of its WSDL file dating from 6/26/2006 to 8/31/2010.
- **FedEx Rate Service**. The FedEx Rate Service operations provide a shipping rate quote for a specific service combination depending on the origin and destination information supplied in the request. We studied 9 versions of this service.
- **FedEx Package Movement Information Service**. The FedEx Package Movement Information Service operations can be used to check service availability, route and postal codes between an origin and destination. We studied 3 versions of this service.
- **PayPal SOAP API**. The PayPal API Service can be used to make payments, search transactions, refund payments, view transaction information, and other business functions. We studied 4 versions of this service.
- **Bing Search Service**. Bing Services provide programmatic access to Bing data by way of application programming interfaces (APIs).The Bing API, Version 2 provides developers and site managers with flexible, multiple-protocol access to content SourceTypes such as Image, InstantAnswer, MobileWeb, News, Phonebook, RelatedSearch, Spell, Translation, Video, and Web. We studied 5 versions of this service.

[6]http://www.r-project.org/

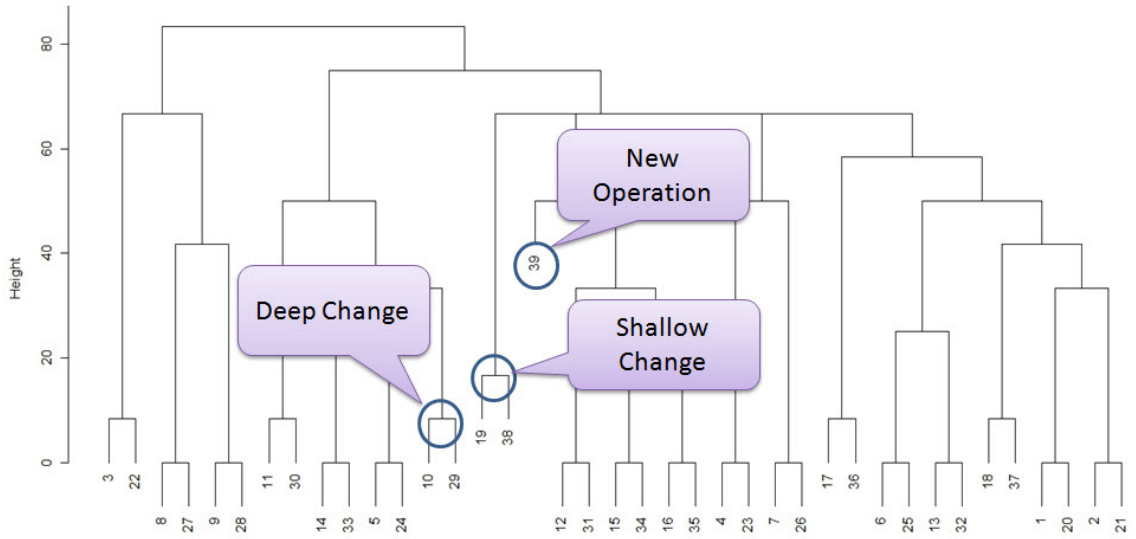


Figure 3. Dendrogram for the clustering of the operations between versions 4 and 5.

A. Analyzing the evolution of the services

Table I shows the evolution profile of all the examined services. The percentage calculated for each one of the activities (change, deletion, insertion) is with respect to the total number of activities in that particular version. As we can see from the table in services like PayPal, Bing and in most versions of Amazon EC2 and FedEx Rate, we observe a domination of additions. From this we can derive two conclusions: (a) these services were in a stage of rapid development and high expansion during this part of their lifecycle and (b) radical changes and deletions are usually avoided because they are more likely to break a client. On the other hand, in services like FedEx Package Movement Information and some versions of the Amazon EC2 and FedEx Rate, we noticed an increased number of changes, primarily, and deletions. This indicates that these services were in a more stable stage and developers performed restructuring and perfective changes.

B. Correlation between changes and business announcements

As web services are an integral part of modern businesses their consistency is bound to be affected by business decisions. In this section, we are trying to correlate changes that happened in the studied services with business announcements of new features.

1) Amazon EC2: In March 2008, Amazon announced[7] new features for static IP addresses, availability zones and user selectable kernels. These changes were already available in version 7 earlier the same year. In August 2008, they announced the Elastic Block Store (EBS) for persistent storage and the changes were incorporated in version 8. In May 2009, they announced the AWS management console, and plans for load balancing, autoscaling, and cloud monitoring services. The changes were incorporated between versions 9 and 13.

2) FedEx Rate: In March 2010, FedEx announced[8] the FedEx Electronic Trade Documents, Shipping Hazardous Meterials, the FedEx Web Integration Wizard, FedEx Freight Rating and enhancements for the FedEx SmartPost. These changes were incorporated between version 6 and 8. In August 2010, they announced more enhancements for the FedEx SmartPost, "Hold at Location" service expansion, new intra-country shipping options and improvements for hazardous material shipping. These changes were introduced in version 9.

3) Bing: In June 2009, Bing launched[9] the Bing Translator and consequently the types TranslationRequest, TranslationResponse and ArrayOfDeepLink were added between version 2.1 and 2.2. In June 2010 Bing was expanded[10] to handle more entertainment-related queries and the enumerations Shopping, QueryAnnotation, Social, Events and RssFeed were added between versions 2.2 and 2.3.

C. Service Change Scenarios

In this section we provide a collection of change scenarios. Some of them have actually occurred in the set of services we have studied, while others we deem possible to happen. We discuss the changes in detail and describe how they can affect client applications: whether they are manageable and how.

[7]Source: Amazon Web Services Blog (http://aws.typepad.com/)

[8]Source: https://www.fedex.com/us/developer/wss/announcement.html

[9]Source: http://en.wikipedia.org/wiki/Bing_Translator

[10]Source:http://blogs.computerworld.com/16374/microsoft_to_add_enhancements_to_bing

Table I
THE EVOLUTION PROFILE OF THE STUDIED SERVICES.

Service	Version	Changed(%)	Deleted(%)	Inserted(%)
Amazon EC2	2	2.82	0	97.18
Amazon EC2	3	13.33	0	86.67
Amazon EC2	4	50	0	50
Amazon EC2	5	8.82	0	91.18
Amazon EC2	6	16.67	50	33.33
Amazon EC2	7	1.71	0	98.29
Amazon EC2	8	1.40	0	98.60
Amazon EC2	9	3.54	0.88	95.58
Amazon EC2	10	11.11	0	88.89
Amazon EC2	11	2.67	0	97.33
Amazon EC2	12	5.56	0	94.44
Amazon EC2	13	0.79	0	99.21
Amazon EC2	14	2.70	0	97.30
Amazon EC2	15	10.26	0	89.74
Amazon EC2	16	1.08	0	98.92
Amazon EC2	17	64.90	0	35.10
Amazon EC2	18	31.06	0	68.94
FedEx Rate	2	8.93	21.43	69.64
FedEx Rate	3	9.20	5.75	85.06
FedEx Rate	4	8.11	17.05	74.84
FedEx Rate	5	8.00	20.00	72.00
FedEx Rate	6	1.51	6.67	91.83
FedEx Rate	7	3.05	30.46	66.50
FedEx Rate	8	11.48	12.02	76.50
FedEx Rate	9	11.53	42.88	45.59
Bing	2.1	0	21.33	78.67
Bing	2.2	0	9.38	90.63
Bing	2.3	0	0	100.00
Bing	2.4	0	0	100.00
PayPal	53.0	2.33	0	97.67
PayPal	62.0	0.55	0	99.45
PayPal	65.1	1.35	0	98.65
FedEx Pack.	3	80.00	0	20.00
FedEx Pack.	4	100.00	0	0

Operation Deletions. We noticed an absence of operation deletions. This is mainly because if an operation is deleted a client that might have been using it will instantly break. This is a non-recoverable situation which in fact means that the client should be changed and recompiled. We found a case like that in FedEx Rate. In version 1 only one operation existed named *getRate*. In version 2 a second operation was added named *rateAvailableServices* and in version 3 a third operation named *getRates* replaced the other two. Although, the three operations were similar in the sense that they used and returned similar data, in the end these changes must have caused significant problems to client applications. A prudent thing to do in this case would be to declare the old operations as deprecated so as not to be used by new clients and when the time was right to remove them in order to minimize the cost.

Inline Type. In Amazon EC2, we noticed the best way to handle changes in types. In version 6 the type RunInstancesInfoType was removed and all of its elements were moved to the parent type named RunInstancesType. This change is called "Inline Type" [4] and it is non-destructive to the client. It does not affect the functionality and it does not break the code because any data that existed in version 5 still exists in version 6 although bundled in a different type. The important thing is that no matter how the data is formatted the client must have access to it because it was used from the previous version. Furthermore, we noticed that old operations never use new data, which is reserved only for the new operations. The opposite action, namely "Extract Type" where a complex type is decomposed in simpler elements is also a recoverable change by the client for the same reasons.

Aggressive Refactoring. An interesting situation occurred in FedEx Rate in version 9 where several enhancements were supposed to take place. For this reason, more than 50% of the types of the service were removed and totally new ones were added. This was a poor maintenance activity not only because of the nature of the changes (deletions) but also their breadth. In this case, the best course of action would be to add the new types in a new service and copy the old still valid components from the previous version and offer both services as alternatives so that the old clients will not break.

Renaming Variables. In Amazon version 14 we had a type with two elements named currentState and previousState. In version 15 the same type has the elements previousState and shutdownState. There are two scenarios in this case. First, that the currentState was renamed in shutdownState and previousState remained the same. In the second case, the currentState was renamed in previousStateand previousState was renamed in shutdownState. The difference between the two scenarios is the order of the parameters. If the order matters then the second scenario is true.

Adding New Types. If new types are added as elements in already existing types then the interface of the service is not affected. The question is then whether the functionality breaks. If the new elements do not participate in the result of the operation then the functionality of the client is not affected. For example, in version 1 of a service we have an operation add(int i, int j) which returns the sum of i and j and in version 2 we have add(int i, int j, boolean flag) where the flag notes whether the result should be stored in a file. In this scenario the flag doesn't affect the sum of the two numbers and thus the functionality of the client will not be affected.

In a different situation, the returned sum will not be the expected one. The problem is that we cannot be sure whether the change in the result was because of the added parameter or because something changed in the algorithm (a change which is not visible to the client). In this case, creating an adapter will not fix the client. An idea to address this problem would be to employ web mining techniques in order to obtain the description of the changes that happened from one version to the other and help the developer of the client

to apply the proper changes in their system.

Changing Input or Output Types. Since the client interacts only with the operations, these are the sensitive points. In order for the interface of the service to break, one has to replace the input or the output types of an operation with different types or rename the input and output types of the operation. In this case the client should update itself in order to invoke the operation in the correct way. If the input or output is replaced by a new type then the type should be generated on the client side or added in the stub.

If the input or output type is changed (elements added, deleted, changed or renamed) then a problem occurs only if the client tries to access these types. For example, if in the returned type of an operation we delete or rename some elements and the client tries to access these elements, it will break. In case of added elements, we will not have problems.

VI. Related Work

A. Model and Tree Differencing Techniques

Fluri et al. [5] proposed a tree differencing algorithm for fine-grained source code change extraction. Their algorithm takes as input two abstract syntax trees and extracts the changes by finding a match between the nodes of the compared trees. Moreover, it produces a minimum edit script that can transform one tree into the other given the computed matching. The proposed algorithm uses the bigram string similarity to match source code statements (such as method invocations, condition statements, and so forth) and the subtree similarity of Chawathe et al. [6] to match source code structures (such as if statements or loops).

Kelter et al. [7] proposed a generic algorithm for computing differences between UML models encoded as XMI files. The algorithm first tries to detect matches in a bottom-up phase by initially comparing the leaf elements and subsequently their parents in a recursive manner until a match is detected at some level. When detecting such a match, the algorithm switches into a top-down phase that propagates the last match to all child elements of the matched elements in order to deduce their differences. The algorithm reports four different types of differences, namely structural (denoting the insertion or deletion of elements), attribute (denoting elements that differ in their attributes' values), reference (denoting elements whose references are different in the two models) and move (denoting the move of an element to another parent element).

Xing and Stroulia [8], [9] proposed the *UMLDiff* algorithm for automatically detecting structural changes between the designs of subsequent versions of object-oriented software. The algorithm produces as output a tree of structural changes that reports the differences between the two design versions in terms of additions, removals, moves, renamings of packages, classes, interfaces, fields and methods, changes to their attributes, and changes of the dependencies among these entities. UMLDiff employs two heuristics (i.e., name-similarity and structure-similarity) for recognizing the conceptually same entities in the two compared system versions. These two heuristics enable UMLDiff to recognize that two entities are the same even after they have been renamed and/or moved. The UMLDiff algorithm has been employed for detecting refactorings performed during the evolution of object-oriented software systems, based on UMLDiff change-facts queries [10].

Recently, Xing [11] proposed a general framework for model comparison, named *GenericDiff*. It tackles the challenge of balancing balance between being domain independent yet aware of domain-specific model properties and syntax by separating the specification of domain-specific inputs from the generic graph matching process and by making use of two data structures (i.e., typed attributed graph and pairup graph) to encode the domain-specific properties and syntax so that they can be uniformly exploited in the generic matching process. Unlike the aforementioned approaches that examine only immediate common neighbors, GenericDiff employs a random walk on the pairup graph to spread the correspondence value (i.e., a measurement of the quality of the match it represents) in the graph.

B. Service Evolution Analysis

Wang and Capretz [12] proposed an impact analysis model as a means to analyze the evolution of dependencies among services. By constructing the intra-service relation matrix for each service (capturing the relations among the elements of a single service) and the inter-service relation matrix for each pair of services (capturing the relations among the elements of two different services) it is possible to calculate the impact effect caused by a change in a given service element. A relation exists from element x to element y if the output elements of x are the input elements of y, or if there is a semantic mapping or correspondence built between elements of x and y. Finally, the intra- and inter service relation matrices can be employed to support service change operations, such as the addition, deletion, modification, merging and splitting of elements.

Aversano et al. [13] proposed an approach, based on Formal Concept Analysis, to understand how relationships between sets of services change across service evolution. To this end, their approach builds a lattice upon a context obtained from service description or operation parameters, which helps to understand similarities between services, inheritance relationships, and to identify common features. As the service evolves (and thus relationships between services change) its position in the lattice will change, thus highlighting which are the new service features, and how the relationships with other services have been changed.

Ryu et al. [14] proposed a methodology for addressing the dynamic protocol evolution problem, which is related with the migration of ongoing instances (conversations) of

a service from an older business protocol to a new one. To this end, they developed a method that performs change impact analysis on ongoing instances, based on protocol models, and classifies the active instances as migrateable or non-migrateable. This automatic classification plays an important role in supporting flexibility in service-oriented architectures, where there are large numbers of interacting services, and it is required to dynamically adapt to the new requirements and opportunities proposed over time.

Pasquale et al. [15] propose a configuration management method to control dependencies between and changes of service artifacts including web services, application servers, file systems and data repositories across different domains. Along with the service artifacts, Smart Configuration Items, which are in XML format, are also published. The SCIs have special properties for each artifact such as host name, id etc. Interested parties (like other application servers) can register to the SCIs and receive notifications for changes to the respective artifact by means of ATOM feeds and REST calls. Using a discovery mechanism the method is able to identify new, removed or modified SCIs. If a SCI is identified as modified, then the discovery mechanism tracks the differences between the two items and adds them as entries in the new SCI. The changes are limited to delete, add, modify a property or delete, add, modify a dependency. The changes are also too general, for example, a change in an input type of an operation is reported as a change in the operations part of a WSDL. In our case, we are more interested in finding more complicated changes and annotate them appropriately so we know the exact nature of the change (where it happened and what it affected).

The aforementioned research works mainly focus on the evolution of inter-dependencies among services or the evolution of business protocols. On the other hand, our approach focuses on the evolution of the elements within a single service and their intra-dependencies. Furthermore, our approach investigates the effect of service evolution changes on client applications.

VII. Conclusion

In this paper we presented an empirical study for the evolution of web services where we investigated how changes that occur in WSDL files can potentially affect client applications. The first question we tried to answer was whether a generic tree differencing algorithm like VTracker can be used to study the evolution of web services. VTracker was indeed helpful in this task mainly because of its ability to produce fine-grained results in terms of distances between individual elements (types and operations) that belonged in different versions and changes that were applied on these elements. This helped us identify the nature of the changes and identify good and bad maintenance scenarios in our case studies. Furthermore, this work helped us improve VTracker in terms of efficiency and accuracy.

For the empirical study we examined the evolution of five web services (Amazon EC2, FedEx Rate, Bing, PayPal and FedEx Package Movement Information). The main conclusion of this study was that web services are usually expanded rather than changed or having their elements removed. This is because the addition of new features does not affect the robustness of clients that already use the service. Furthermore, changes, if made in a conservative manner, do not affect clients much. On the other hand, deletion of elements should be avoided in all cases as it can easily break a client application.

Acknowledgment

The authors would like to acknowledge the generous support of NSERC, iCORE, and IBM.

References

[1] R. Mikhaiel, G. Lin, and E. Stroulia, "Simplicity in RNA Secondary Structure Alignment: Towards biologically plausible alignments," *6th IEEE Symposium on Bioinformatics and Bioengineering*, 2006.

[2] K. Zhang, R. Stgatman, and D. Shasha, "Simple fast algorithm for the editing distance between trees and related problems," *SIAM Journal on Computing*, vol. 18, pp. 1245–1262, 1989.

[3] R. Mikhaiel and E. Stroulia, "Examining Usage Protocols for Service Discovery," *4th International Conference on Service Oriented Computing*, pp. 496–502, 2006.

[4] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, *Refactoring Improving the Design of Existing Code*. Boston, MA: Addison Wesley, 1999.

[5] B. Fluri, M. Würsch, M. Pinzger, and H. C. Gall, "Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction," *IEEE Transactions on Software Engineering*, vol. 33, no. 11, pp. 725–743, 2007.

[6] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change Detection in Hierarchically Structured Information," *ACM Sigmod Internation Conference on Management of Data*, pp. 493–504, 1996.

[7] U. Kelter, J. Wehren, and J. Niere, "A Generic Difference Algorithm for UML Models," *Software Engineering 2005, Fachtagung des GI-Fachbereichs Softwaretechnik*, pp. 105–116, 2005.

[8] Z. Xing and E. Stroulia, "UMLDiff: An Algorithm for Object-Oriented Design Differencing," *20th IEEE/ACM International Conference on Automated Software Engineering*, pp. 54–65, 2005.

[9] ——, "Analyzing the Evolutionary History of the Logical Design of Object-Oriented Software," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 850–868, 2005.

[10] ——, "Refactoring Detection based on UMLDiff Change-Facts Queries," *13th Working Conference on Reverse Engineering*, pp. 263–274, 2006.

[11] Z. Xing, "Model Comparison with GenericDiff," *25th IEEE/ACM International Conference on Automated Software Engineering*, pp. 135–138, 2010.

[12] S. Wang and M. A. M. Capretz, "A Dependency Impact Analysis Model for Web Services Evolution," *IEEE International Conference on Web Services*, pp. 359–365, 2009.

[13] L. Aversano, M. Bruno, M. D. Penta, A. Falanga, and R. Scognamiglio, "Visualizing the Evolution of Web Services using Formal Concept Analysis," *8th International Workshop on Principles of Software Evolution*, pp. 57–60, 2005.

[14] S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul, "Supporting the Dynamic Evolution of Web Service Protocols in Service-Oriented Architectures," *ACM Transactions on the Web*, vol. 2, no. 2, pp. 1–46, 2008.

[15] L. Pasquale, J. Laredo, H. Ludwig, K. Bhattacharya, and B. Wassermann, "Distributed cross-domain configuration management," in *Proceedings of the 7th International Joint Conference on Service-Oriented Computing*, ser. ICSOC-ServiceWave '09, 2009, pp. 622–636.

2.2 WSMeta: a meta-model for web services to compare service interfaces

Fokaefs, M., Stroulia, E., 2013b. WSMeta: a meta-model for web services to compare service interfaces. In: Panhellenic Conference on Informatics (PCI 2013). ACM, pp. 1-8.

WSMeta: A Meta-Model for Web Services to Compare Service Interfaces

Marios Fokaefs and Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
{fokaefs,stroulia}@ualberta.ca

ABSTRACT

With the increasing adoption of the web-services stack of standards, service-oriented architecture has attracted substantial interest from the research community which has produced several languages and methods for describing and reasoning about services. These languages cover many concepts ranging from individual services and their code generation from specifications, service semantics, service compositions and networks, economics and business aspects around service ecosystems etc. However, this abundance of specification languages has also resulted in communication difficulties between stakeholders and hinders tasks such as service composition, discovery and maintenance. The presented work is a step towards the unification of the specifications and different aspects of service systems using Model-Driven Engineering. We propose a generic and abstract web service meta-model called WSMeta, which has the ability to describe both operation-centric web services (WS-*) and data-centric web services (REST) and can be used in tasks such as service evolution analysis and service systems maintenance.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*web-based services*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*modules and interfaces, top-down programming*

General Terms

Design

Keywords

model-driven engineering, service-oriented architectures, WADL, WSDL, service comparison

1. INTRODUCTION

PCI 2013 September 19 - 21 2013, Thessaloniki, Greece

http://dx.doi.org/10.1145/2491845.2491860 ...$15.00.

In spite of being a relatively new technology, web services have been extensively studied by academia and industry alike. All this research effort has resulted in the development of a variety of specifications for web services and service systems. Oberle [11] gives a comprehensive overview of these specifications highlighting their purposes and their properties. One of the reasons that contributed to the development of multiple service-related specifications is the dichotomy between operation-centric and data-centric web services. Instances in the former category are usually specified using the Web Service Description Language (WSDL)[1], a W3C standard. WSDL files are XML-based and specify the complete interface of a web service in terms of operations and the data types they manipulate as input and output; essentially a WSDL specification serves (a) as a directory of the operations supported by the service, and (b) as a complete guideline for how clients can invoke the service. A variety of supporting tools have been developed to generate client- and server-side code (in a variety of programming languages) from WSDL specifications, or to reverse engineer WSDL specifications from code. Data-centric web services conform to the Representational State Transfer (REST) [7] architectural style, and they are usually specified in non-standard HTML or XHTML documents. In this informal specifications natural-language text is a dominant element, thus, lacking structure which in turn makes it difficult for the specification to be parsed automatically. For this reason, a new specification has been submitted recently to W3C for standardization called Web Application Description Language (WADL)[2]. WADLs are also XML-based, in the spirit of WSDL, and can be used for client code generation or client configuration. They describe data as resources and all the operations that can be invoked on these resources in the form of HTTP methods (GET, POST, PUT, DELETE).

Services may follow either the REST or the WS-* style and there is often a need for instances in either style to interoperate. To that end, there have been tools and techniques to translate service specification from one format to the other. Furthermore, version 2.0 of the WSDL standard has been designed so that it can be used to natively specify both REST and WS-* services (i.e., there is no need for intermediate transformations). However, in reality, both formats are used independently and this can raise certain challenges. For example, if one wants to create a service composition, one should be able to take full advantage of both REST and WS-* repositories alike. This is why, we propose a web ser-

[1]http://www.w3.org/TR/wsdl20/
[2]http://www.w3.org/Submission/wadl/

vice meta-model called *WSMeta*. This meta-model is simply an abstraction of the WADL and WSDL specifications. It takes into account all the common elements between the two and eventually specifies a service as a collection of operations with inputs and outputs. The meta-model also contains extension points as place-holders for other concepts of service systems such as clients and service compositions. WSMeta is based on Ecore, the Eclipse Modelling Framework (EMF)[3] meta-model. We also provide a set of Epsilon Transformation Language (ETL)[4] scripts to transform WSDL and WADL files into a WSMeta model and vice versa.

WSMeta can be primarily useful to service consumers. The task of service interface comparison is very common for service consumers. This task is performed, for example, when the consumed service has changed and the consumer has to compare the two versions of the interface in order to identify the changes and adapt to the new version. In our previous work [8] we have investigated the evolution of web services and we have identified certain types of changes and how they can affect client applications. The task of differencing web services requires the parsing of the service specification and comparison of its elements between different versions. However, certain properties of the service may be irrelevant to the task at hand, for instance, namespaces, unused operations or types and so on. In this case, a lightweight model to represent the interfaces, like WSMeta, can improve the accuracy and the efficiency of the comparison method. Another scenario where comparison is necessary is when the client decides to migrate the application to the service of another vendor. In this case, the client needs to compare the interfaces of the two services and identify the necessary changes to the client application so that it can be migrated to the new service. A generic web service meta-model, like WSMeta, can allow the client to compare web services, whose interfaces are specified in different standards, for example a WSDL service with a RESTful service. This way, the client's possibilities are greatly expanded. Finally, WSMeta can also be useful to service providers. The meta-model allows the provider to change the format of a service (from WSDL to REST or vice versa) or offer the same service in different formats, which will enable the provider to expand the clientèle and accommodate more applications. The model transformation that accompany WSMeta are specific to service providers for this particular use case.

The rest of the paper is outlined as follows. Section 2 provides an overview of other model-driven approaches that have been applied on Web Services. Section 3 describes WSMeta and provides details on how the meta-model was created and in Section 4 we discuss how the transformation can be applied on a simple service. Finally, Section 5 concludes this work and discusses some of our future plans.

2. RELATED WORK

2.1 Representing service systems

Ortiz and Hernandez [12] propose a model-driven approach to develop web services with extra-functional properties from a platform independent model (PIM). As a PIM, they use a UML profile extended with certain stereotypes for service components and extra-functional properties. Then, using ATL (Atlas Transformation Language) [5] transformation rules this model is transformed into four platform specific models, each serving a certain purpose:
(a) a web service meta-model in JAX-RPC [6], in order to decouple the properties from the service development process;
(b) an aspect-oriented meta-model, in order to decouple the properties from the implementation of the service;
(c) a policy meta-model in WS-Policy, in order to decouple the properties from the description of the service; and finally
(d) a SOAP tag meta-model, to make the services more flexible in the presence of optional extra-functional properties. The amalgamation of model-driven engineering, service component architecture, aspect-orientation and WS-Policy, as presented in the paper, is exemplified in a simple case study.

Jegadeesan and Balasubramaniam [10] present a service meta-model, which extends the UML Infrastructure Library. The motivation behind this work, is the inability of existing modelling techniques to capture all aspects of SOA and the evolution of web service standards. The proposed meta-model has five views each one capturing an aspect of SOA:
The *Service Definition View* captures ownership information as well as the nature of the service (composite, abstract etc.);
The *Service Capability View* captures the description of a service and its operations, QoS properties, constraints and exceptions;
The *Service Policy View* captures the non-functional constraints of a service;
The *Service Realization View* captures information about how a service is realized such as participants, like providers, consumers, aggregators and mediators, and how services can be composed.
The *Service Mediation View*, that handles compositions from a data or process perspective.

Treiber et al. [14] propose SEMF, a service evolution management framework. According to the authors, different stakeholders of a web service system that handle the various aspects or artifacts of the system can trigger various changes. The proposed framework handles all this data and integrates it by employing a Web Service Information Model and then correlates the various changes from the different stakeholders. The information model is implemented as Atom feeds, where the artifacts are linked seamlessly regardless of the underlying data model and the feeds can be queried efficiently using XQuery. The framework is also extendible to accept new data sources and the extensions are in the form of plugins which follow a specified interface.

Fensel and Bussler [6] present a Web Service Modelling Framework (WSMF) that focuses particularly on supporting e-commerce applications with multiple and heterogeneous partners. With respect to the service system implementation, the framework focuses on supporting service composition and mediation between the partners. According to the authors such a modelling framework should satisfy two fundamental properties that need to exist in an e-commerce system: *decoupling* of the system's parts and *mediation* between the various partners of the system. WMSF provides support for discovering, comparing and composing web services and handling heterogeneous data and business logics.

[3]http://www.eclipse.org/modeling/emf/
[4]http://www.eclipse.org/gmt/epsilon/doc/etl/
[5]http://www.eclipse.org/atl/
[6]http://java.net/projects/jax-rpc/

WSMF uses four elements to describe e-commerce systems. Ontologies are employed to provide domain and application-specific terminology needed by the system. A goal repository contains the various objectives of the partners and their specific pre- and post-conditions. The web services used in the system are characterized by their individual descriptions. Finally, mediators are defined for eliminating discrepancies between data structures, business logics, message-exchange protocols, dynamic service invocation and service composition.

Finally, Andrikopoulos et al. [3] follow a model-based approach to facilitate the management of service evolution. They propose three different descriptions for a service system: (a) the Abstract Service Definition (ASD) that describes the general concepts and their relations that are common to all web services; (b) the Service Schema Definition (SSD) that describes a particular web service; and (c) the Instance of Service Schema Definition (ISD) that describes the execution of a web service. In fact, the ASD is the meta-model of the SSD. As such, it contains attributes which are assigned values when the service is instantiated (i.e., in the ISD) and property domains from which a particular property will be selected when a SSD will be generated from the ASD. The ASD comprises of two sections, the public and the private and three layers, the structural, the behavioural and the regulatory. The public section refers to the public interface of the web service (i.e., the one that is exposed to potential clients) and the private section refers to the internal logic of the service. The structural layer contains the interface specific components of the service such as operations, messages and data types, the behavioural layer contains constraints and dependencies between the operations, as well as instructions on how the operations can be combined into a process and, finally, the regulatory layer contains the policies and the business rules of the service.

2.2 Task- or component-specific service models

Cao et al. [5] first recognize the need for a web service meta-model. According to the authors, the meta-model should be a platform-independent model (PIM) in order to be generic). This meta-model can later be transformed into a platform-specific model (PSM). In this work, the proposed tool-independent meta-model is realized using ER diagrams, while the tool-dependent model is specified using Generic Modelling Environment. This approach covers only interface specific elements.

Bordbar and Staikopoulos [4] propose a semi-automatic method to generate meta-models from XSD files. They use hyperModel, an Eclipse plug-in to generate XMI files from XSD. Next, using Poseidon for UML they generate a UML class diagram. According to the authors, the proposed methodology is especially useful for web services since WSDL files are practically XSD based. They demonstrate this ability by generating a WSDL meta-model from the WSDL XSD published by W3C and manually refining it to remove XSD or XML specific elements leaving behind only pure WSDL elements. They also propose a meta-model approach to support integration and interoperability between the different aspects and standards of web-service systems [13], based on UML-related research, where model transformations to allow for integration and interoperability are already established. First, they discuss the mechanisms from both UML and web services and compare them based on five criteria: containers, composite structures, messaging, links and mechanisms. Next, they propose a binding meta-model for web-service integration and interoperability and they explain how this meta-model can be applied to integrate BPEL compositions with WSDL specifications.

Ali and Babar [1] propose an extension for SoaML called AmbientSoaML to incorporate mobility properties, such as boundaries in mobile networks, in the SoaML meta-model, using Ambient Calculus. The problem is motivated and the method is explained in a simple example about voice calling and sms services in mobiles. Later, Ali et al. [2] present an Eclipse plug-in for developing web services in SoaML[7]. They use SoaML as their PIM. The meta-model is specified in Ecore and the users can create instances of it using EMF. The instances can be validated against the meta-model using the EMF Validation Framework. The plug-in also provides a SoaML Graphical Editor to create and edit SoaML instances. Finally, using ATL the SoaML models can be transformed into OSGi Declarative Service models.

Gebhart et al. [9] propose a set of properties for designing service systems based on SoaML. For properties that can be quantified OCL expressions are used to evaluate such properties. Given the values of these properties, a design can be characterized as bad and a better one can be suggested, as demonstrated in the paper.

2.3 Discussion

Although, as we see there have been proposed several model-driven engineering techniques to specify web services and service systems, they don't satisfy the entirety of our needs with respect to the particular task we try to address (i.e., comparison of heterogeneous web service specifications). In this work, the proposed meta-model is not intended to be used as a means to specify new services or service systems, but rather to be used as an intermediate format to be used in tasks that include comparison of service interfaces such as evolution analysis, service discovery and selection, etc.

While holistic approaches may cover many aspects of a service system such as partners, policies, business logic etc., these approaches may not provide sufficient details for these parts. For example, in SoaML, the interface of a web service is implemented as a single component. However, in a task like comparison, one needs to specify all the interface components (operations, messages and types) as individual components and analyse them separately.

An advantage of holistic approaches, especially the ones that are based on established modelling languages and techniques like UML, is that they provide opportunities for extension. Developers can use UML stereotypes, OCL rules and other components to extend a service model and include more details or other aspects. In this sense, one can extend a service model with a more fine-grained model for the service interface. Unfortunately, most of the approaches lack the tool support and detailed instructions on how to extend the models so that the extension communicates harmoniously with the core model.

3. WSMETA

Before embarking into the task of creating a web service meta-model, we first need to identify the elements that the

[7]http://www.omg.org/spec/SoaML/

two main formats that we want to integrate (WADL and WSDL) have in common. Therefore, a pre-processing step would be to analyse the two formats so that we can understand the purpose and the significance of each element. Then, we produce a mapping between the two formats. In some cases, elements in one format may correspond directly to elements in the other. In other cases, the elements of one format might be more complex and correspond to a collection of elements in the other format. Finally, we include in the meta-model the elements that are not common in the two formats, since, when translating from a specific format to WSMeta, we need to keep all the important information so that we can later perform the opposite translation. In this section, we will present these steps of the WSMeta construction in detail.

3.1 Analyzing WSDL

As we have already mentioned, WSDL specifies a service as a collection of operations that require an input and return an output or throw exceptions. A WSDL file consists of three sections: the schema that specifies the data types, the interface that specifies the operations, and the binding that specifies implementation-related aspects of the service. The schema is usually specified as an XML Schema Document (XSD), that can be included, imported or specified as an inline document inside the WSDL. It describes the data that the service handles and organizes it in complex or simple types. The operations are identified by a name and contain an input, an output, a fault corresponding to the input and a fault corresponding to the output. The number and types of these children are dictated by the message-exchange pattern of the operation (request-response, notification etc.). Inputs, outputs and faults have a reference to a type from the schema. The binding of the service specifies how the service is bound to a communication protocol (e.g. RPC/encoded, Document/literal etc.). In the binding, one can also specify what is the communication protocol over which an operation can be invoked. For example, we can have `wsoap:action` and the name of an operation for SOAP or `whttp:method` and an HTTP method for HTTP. Finally, it also contains the service's address.

3.2 Analyzing WADL

WADL describes a service as a set of *resources* on which operations can be applied as HTTP methods (GET, PUT, POST, DELETE). As such, the WADL can be split in two parts: the schema and the resources. The schema serves the same purpose as in WSDL, although in WADL it is referred to as a grammar. There is no restriction on what format the schema is specified, but XSD is a popular choice in WADL as well.

The resources part includes a URL that is the path to the resources. Each resource has a special attribute that defines the path to this particular resource. To find the actual address to a resource one has to concatenate its path attribute with the URL to the resources. Each resource contains the methods that are valid to be applied on it. The method is identified by an id and the name of an HTTP method. A method contains a request and multiple responses depending on the HTTP status (by default, WADL methods follow the request-response message exchange pattern [8]). For example, an HTTP 200 status code means success while 400 means failure. Responses and requests can be specified in two ways: either as representations or as a set of parameters. A representation specifies the type of the media, which can be, for example, XML or JSON, and the structure of the data, which can be specified as an XSD type or it can be a reference to an XSD type declared in the grammars section. A parameter has a name and a reference to an XSD type specified in the grammars. It also has other attributes such as whether it is required or a default value if necessary. Parameters can also be enumerations of options.

3.3 Mapping WADL and WSDL

In order to map the elements and their attributes between WADL and WSDL, we used VTracker, a tree-aligning algorithm, which we have already used to analyse the evolution of WSDL services [8]. VTracker has a special feature that allows us to create a cost function for two different file formats. Essentially, we ran VTracker on the WSDL and WADL schemas and we produced a mapping between the elements of the two standards with their corresponding distances. Because VTracker employs not only the structural properties of the elements but also their references (both incoming and outgoing), even in the case of elements that are different structurally, the algorithm can map them based on what other elements they use and by what other elements they are used. The synthesized cost function helped us map several elements. However, there were cases where not even references helped and for specific elements there was no clear mapping between the two formats. In these cases, we specified a mapping manually by changing the distance between the two elements to 0 in the cost function. VTracker was selected due to its bootstrapping ability to calculate a cost function by comparing a schema to itself or two different schemas. Although this cost function might require some manual tuning, the bootstrapping process performs most of the mapping automatically. In any case, the manual effort is only applied on the schema level and the cost function is used for instance level comparisons.

In order to evaluate the mapping, we created a reduced version of the Amazon EC2[9] service in WSDL, by including only one operation and its corresponding types. Then, we manually translated this service to WADL. The next step was to compare the two formats with VTracker by using various configurations of the algorithm. The first variability point we introduced concerned the employed cost function. We used three configurations: no special cost function, the synthesized cost function, and the manually improved cost function. The second variability point concerned the key elements. VTracker requires from the developer to specify what are the key elements in each file (i.e. the elements for which we are interested in mapping between the two compared files). Then the algorithm reports the mapping between the key elements. If an element is not considered key, it does not participate in the mapping process. First, we included XSD elements, WSDL operations and WADL methods as key elements. Next, we introduced some "noise" in the key elements by adding WSDL messages, WSDL ports and WADL resources. With the first key configuration, all cost functions found the proper mapping (between XSD elements and between WSDL operations and WADL methods). However, the distance reported between the elements (i.e., the confidence of the mapping) was better with the synthe-

[8]http://www.w3.org/2002/ws/cg/2/07/meps.html

[9]http://aws.amazon.com/ec2/

Table 1: Mapping of elements and their attributes between WADL and WSDL.

WADL	WSDL	WSMeta
application	definitions	IService
	@targetnamespace	@targetNamespace
grammars	types	Schema
resource	interface	Interface
resources@base+ resource@path	service::endpoint @address	@address
method	operation	Operation
@id	@name	@name
@name	binding::operation @whttp:method OR binding::operation @wsoap:action	@method
`http://www.w3.org/ns/wsdl/in-out` (fixed)	@pattern	@pattern
@href		@href
	@safe	@safe
	@style	@safe
request (response)	input(output)	Operation::request (response)
::param@type OR ::representation@element	@element	@request(@response)
param	xsd:element	IType
@type	@type	@name
param	xsd:simpleType	SimpleType
::option@value	::restriction::enumeration @value	Option@value

sized cost function than with no special cost function and even better with the manually improved cost function. With the second key configuration and with the default cost function and the synthesized cost function, VTracker got confused and mapped WADL methods with WSDL messages. This happened because WADL methods can reference XSD elements directly (if they use parameters instead of representations). WSDL operations can only access XSD elements only through messages (at least in version 1.1[10]). Therefore, it seems more natural to map such WADL methods with WSDL message since they refer to the same elements and their structure is not very different. However, with the manually improved cost function, where we explicitly specified that WSDL operations are to be mapped with WADL methods, the mapping was correct and with a strong confidence.

Table 1 shows the final mapping between WADL and WSDL. We use the delimiter '::' to denote a *parent-child* relationships, the delimiter '@' to denote an attribute and the delimiter ':' to qualify names with namespaces. The first column of each specification contains their high level elements, while the second column contains the attributes of these elements.

3.4 The Web Service Meta-Model

Figure 1 shows the proposed web service meta-model, WSMeta, produced by merging the WSDL and WADL models as explained previously. WSMeta was implemented using the EMF Ecore. An interesting point about the meta-model concerns the schema. Although a schema is considered a separate file, it is an essential part of a service since it specifies the nature and the structure of the data the service handles. As such it should be included in a web service meta-model and it can be useful in tasks such as service comparison.

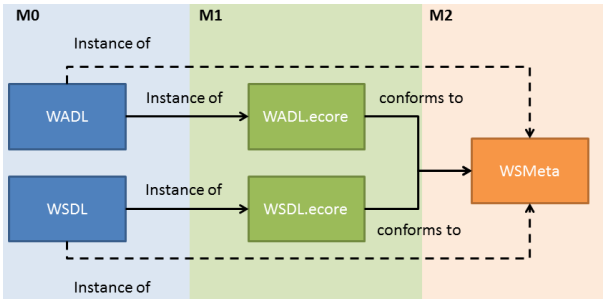


Figure 2: The WSMeta as a meta-model for specific services.

Figure 2 positions WSMeta with respect to specific models. As its name implies, WSMeta is a meta-model for the specific web service models, which have to conform to it. In order to get the model corresponding to a web service file or generate the service from the model, we use the Eclipse Web Standard Tools (WST)[11]. Unfortunately, WST only offers transformations from WSDL to its corresponding model and not from WADL services (as of the time this paper was being written).

Apart from the meta-model, we also contribute a set of transformations, implemented using the Epsilon Transformation Language (ETL). An example of such a transformation rule can be seen in Figure 3. These transformations are

[10]In version 1.2 of WSDL, the operations can directly access data types, therefore the mapping could have been easier in this case

[11]http://www.eclipse.org/webtools/wst/main.php

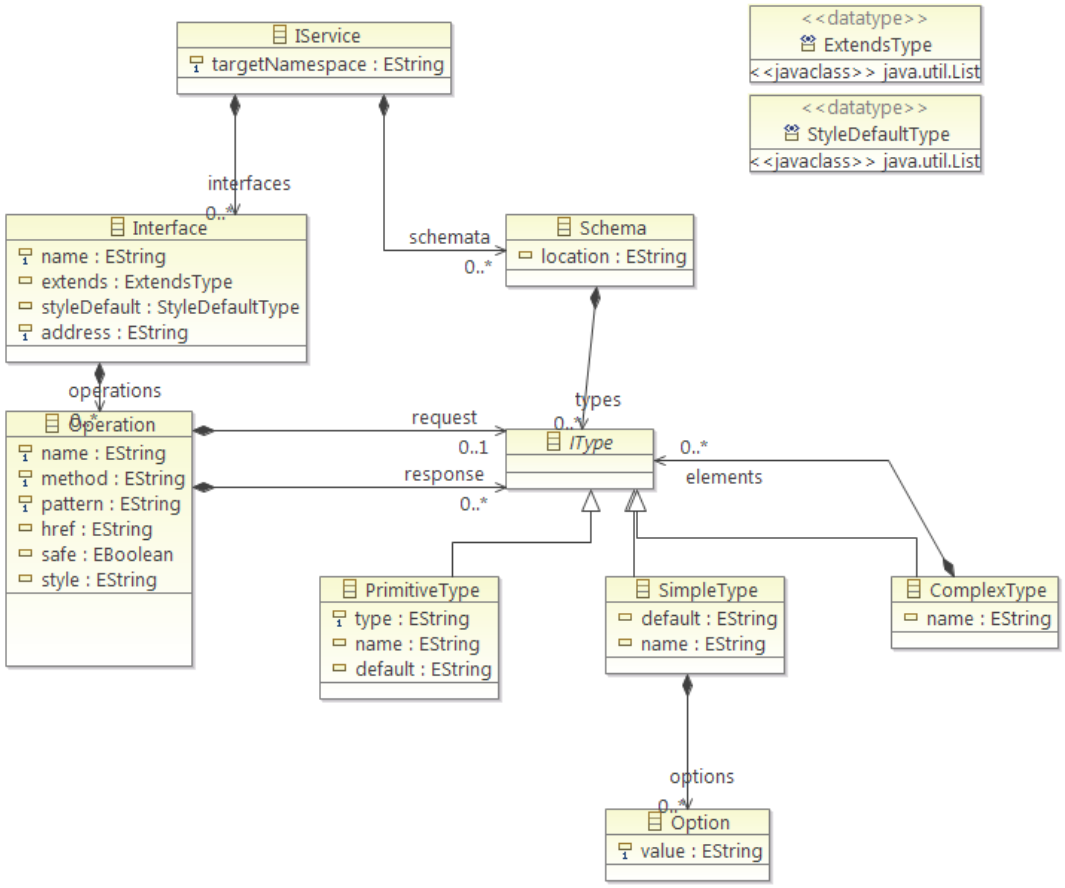


Figure 1: The Web Service meta-model.

```
rule CopyMethodType
    transform s : wadl!MethodType
    to t : wsmeta!Operation {


    t.href := s.href;

    t.name := s.id;

    t.method := s.name;

    t.pattern := "http://www.w3.org/ns/wsdl/in-out";

    t.input ::= s.request;

    t.output ::= s.response;
}
```

Figure 3: ETL rule to transform a WADL method to a WSMeta operation.

used to translate the specific models into the meta-model and vice versa. This way we can easily create meta-model instances from specific web services.

This would also allow us to translate the specification of a service from one standard to the other through WSMeta. The way we have defined our meta-model, i.e., by including not only the common elements between WSDL and WADL but also the unique elements from each standard, we ensure that such a transformation will be lossless. In other words, the produced interface will exactly correspond to the interface of the other standard. Of course, we cannot guarantee that the produced interface will be immediately functional and this is because, for a translated service to work, some low-level requirements need to be fulfilled first. For example, some REST services may require the existence of a persistent resource (e.g., database or file system). If we translate a WSDL interface to a WADL one, the service will not work until such a resource is provided. The purpose of the translation of services using WSMeta is to guide and facilitate the migration of services from one standard to the other. Manual changes to the service interface and the underlying system may still be necessary for the service system to function properly.

One thing that can hinder the transformation of web service interfaces is the schema. Whether the interface includes or imports an XML schema, it is still a different file, not included in the model. This means that for a single interface two models need to be created: one for the functional parts and one for the data types of the service. As a result, the transformation scripts for these two models need to be post-processed in order to resolve the references between the two models.

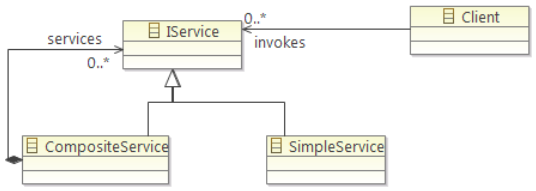


Figure 4: The client/composition extension of WSMeta.

Built with popular and state-of-the-art modelling techniques and tools, WSMeta can be easily extended to include more SOA concepts. For example, Figure 4 shows a WSMeta extension to describe service compositions and to include client applications.

4. AMAZON EC2 CASE STUDY

In this section, we present how we applied the transformation scripts and WSMeta on a case study we created from the Amazon EC2 service. We created a minimal interface with only two operations (`RegisterImage` and `DeregisterImage`) and information only about the abstract part of the interface, without the schema or the binding. We first created a WADL and a WSDL instance of the interface based on the WADL and WSDL schemata respectively. Then using the ETL transformations we generated the WSMeta instances from the other two instances. Finally, using the opposite transformation rules, we attempted to reconstruct the WADL and WSDL instances from the WSMeta instance. The results of this case study are shown in Figures 5 and 6 respectively for the two formats.

As it can be seen from the figures, the transformations from the specific model instance to the meta-model instance and vice versa produced the same model instance as the original in terms of structure. This demonstrates the ability of the meta-model and the transformation scripts to retain the basic structure of the service, which will allow more accurate comparisons. However, there are some discrepancies with respect to specific attributes of the service elements (e.g. names, types etc.), which shows that the transformations are not completely lossless and additional work needs to be done in this aspect. Furthermore, the transformation from both WADL and WSDL to WSMeta produced the same instance in terms of structure, which demonstrates the ability of our method to allow comparisons between heterogeneous service interfaces.

5. CONCLUSION AND FUTURE WORK

In this work, we presented *WSMeta*, a web-service meta-model. We applied VTracker on WSDL and WADL schemas in order to identify the common elements between the two standards. We used Ecore as our modelling language and ETL for the model transformations.

The proposed meta-model is not intended to describe web services but rather to be used as intermediate format to run tasks such as web service interface comparison. Since interfaces are the consumable part of a service, its critical parts are what it can do (operations) and on what data (types). By keeping only these elements, we abstract it and make it more lightweight to facilitate comparison algorithms and improve their efficiency and accuracy. Furthermore, by combining the elements of WSDL and WADL in our meta-model we can compare heterogeneous services specified in these standards. This widens the possibilities in web service discovery and selection. Finally, by using popular and state-of-the-art tools (Ecore, EMF, Epsilon) to create WSMeta and provide tooling support for it, we facilitate its further development and make it easier for other developers to provide custom extensions.

This work is a first step towards a web service meta-model for service interface comparison. Although some of its fundamental aspects have been implemented and presented in this work, we want to further develop and improve WSMeta. First, we plan to systematized the process of creating the meta-model. This can be achieved by model merging techniques. Using such techniques in combination with VTracker, we will no longer have to rely on manual effort to map elements between different formats. Furthermore, we will be able to study more service standards other than WSDL or WADL and automatically integrate them with WSMeta. Finally, Eclipse WST allows us to parse a WSDL service and generate the corresponding Ecore model. However, such a parser doesn't exist for WADL or other standards. It is necessary to create such a tool to allow us to apply the transformations and translate WADL services to WSMeta models.

6. ACKNOWLEDGMENTS

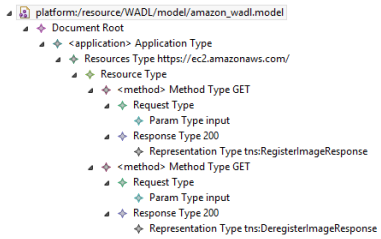
This work has been supported by NSERC, AITF and IBM.

7. REFERENCES

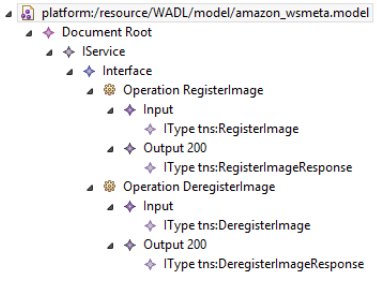
[1] N. Ali and M. A. Babar. Modeling service oriented architectures of mobile applications by extending soaml with ambients. In *Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications (Patras, Greece, August 27-29, 2009)*, SEAA '09, pages 442–449, 2009.

[2] N. Ali, R. Nellipaiappan, R. Chandran, and M. A. Babar. Model driven support for the service oriented architecture modeling language. In *Proceedings of the 2nd International Workshop on Principles of Engineering Service-Oriented Systems (Cape Town, South Africa, May 1-2, 2010)*, PESOS '10, pages 8–14, 2010.

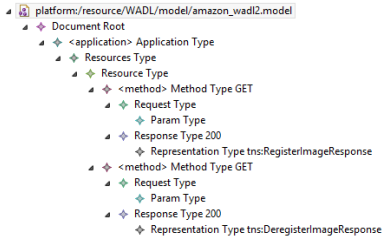
[3] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. Managing the evolution of service specifications. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering (Montpellier, France, June 16-20, 2008)*, CAiSE '08, pages 359–374, 2008.



(a) The original instance of the WADL Amazon service.

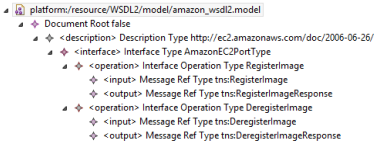


(b) The WSMeta Amazon instance after the WADL2WSMeta transformation.

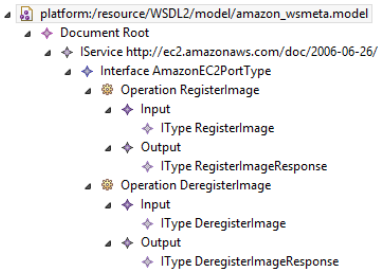


(c) The WADL Amazon instance after the WSMeta2WADL transformation.

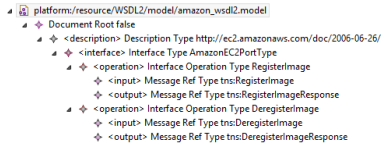
Figure 5: The WADL to WSMeta Amazon case study.



(a) The original instance of the WSDL Amazon service.



(b) The WSMeta Amazon instance after the WSDL2WSMeta transformation.



(c) The WSDL Amazon instance after the WSMeta2WSDL transformation.

Figure 6: The WSDL to WSMeta Amazon case study.

[4] B. Bordbar and A. Staikopoulos. Automated generation of metamodels forweb service languages. In *Proceedings of the 2nd European Workshop on Model Driven Architecture (Canterbury, UK, September 7-8, 2004)*, MDA'04, September 2004.

[5] F. Cao, B. R. Bryant, W. Zhao, C. C. Burt, R. R. Raje, A. M. Olson, and M. Auguston. A meta-modeling approach to web services. In *Proceedings of the IEEE International Conference on Web Services (San Diego, CA, USA, July 6-9, 2004)*, ICWS '04, pages 796–800, 2004.

[6] D. Fensel and C. Bussler. The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[7] R. T. Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures.* Doctoral dissertation, University of California, Irvine, 2000.

[8] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau. An empirical study on web service evolution. In *Proceedings of the 2011 IEEE International Conference on Web Services (Washington, DC, USA, July 4-9, 2011)*, pages 49–56, 2011.

[9] M. Gebhart, M. Baumgartner, S. Oehlert, M. Blersch, and S. Abeck. Evaluation of service designs based on soaml. In *Proceedings of the 2010 Fifth International Conference on Software Engineering Advances (Nice, France, August 22-27, 2010)*, ICSEA '10, pages 7–13, 2010.

[10] H. Jegadeesan and S. Balasubramaniam. An MOF2-based Services Metamodel. *Journal of Object Technology*, 7(8):71–96, 2008.

[11] D. Oberle. D1 report on landscapes of existing service description efforts. `http://www.w3.org/2005/Incubator/usdl/wiki/D1`, September 2011.

[12] G. Ortiz and J. Hernandez. A case study on integrating extra-functional properties in web service model-driven development. In *Proceedings of the Second International Conference on Internet and Web Applications and Services (Mauritius, May 13-19, 2007)*, pages 35–40, 2007.

[13] A. Staikopoulos and B. Bordbar. A comparative study of metamodel integration and interoperability in uml and web services. In *Proceedings of the First European conference on Model Driven Architecture: foundations and Applications (Nürnberg, Germany, November 7-10, 2005)*, pages 145–159, 2005.

[14] M. Treiber, H.-L. Truong, and S. Dustdar. Semf - service evolution management framework. In *Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications (Parma, Italy, September 3-5, 2008)*, SEAA '08, pages 329–336, 2008.

2.3 WSDarwin: Studying the Evolution of Web Service Systems

Fokaefs, M., Stroulia, E., 2014b. WSDarwin: Studying the Evolution of Web Service Systems. Advanced Web Services. Springer, Ch. 9, pp. 199-223.

WSDARWIN: Studying the Evolution of Web Service Systems

Marios Fokaefs and Eleni Stroulia

Abstract The service-oriented architecture paradigm prescribes the development of systems through the composition of services, *i.e.,* network-accessible components, specified by (and invoked through) their interface descriptions. Systems thus developed need to be aware of changes in, and evolve with, their constituent services. Therefore, the accurate recognition of changes in the specification of a service is an essential functionality in supporting the software lifecycle of service-oriented systems. In this chapter, we extend our previous empirical study on the evolution of web-service interfaces and we classify the identified changes according to their impact on client applications. To better understand the evolution of web services, and, more importantly, to facilitate the systematic and automatic maintenance of web-service systems, we introduce WSDARWIN, a specialized differencing method for web services. Finally, we discuss the application of such a comparison method on operation- (WSDL) and resource-centric (REST) web services.

1 Introduction

Service-system evolution and maintenance is an interesting variant of the general software-evolution problem. The problem is complex and challenging due to the fundamentally distributed nature of service-oriented systems, whose constituent parts may reside on different servers, across organizations and beyond the domain of any individual entity's control. At the same time, since the design of a service-oriented system is expressed in terms of the interface specifications of the under-

Marios Fokaefs
Department of Computing Science, University of Alberta, Edmonton, AB, Canada, e-mail: fokaefs@ualberta.ca

Eleni Stroulia
Department of Computing Science, University of Alberta, Edmonton, AB, Canada e-mail: stroulia@ualberta.ca

lying services, the overall system needs and can be aware only of the changes that impact these interface specifications; any changes to the service implementations that do not impact their interfaces are completely transparent to the overall system. In effect, the specifications of the system's constituent services serve as a boundary layer, which precludes service-implementation changes from impacting the overall system.

The directly affected party in the evolution of service systems is the client, *i.e.,* the consuming party. Figure 1 shows a typical evolution scenario from the client's perspective. Initially, the client invokes the service and a fault may be detected. It is not usual for the client to have a priori knowledge about any changes on the service, unless there is frequent and effective communication between the provider and the client. Once the fault is detected, the client has to compare the old service interface with the new one from the provider to identify the nature of the changes and possibly their effect on the application. The next step is to adapt the client application to the new version of the service. This requires as much information as possible in order to make the adaptation process systematic and, if possible, fully automatic. Finally, the client has to test the application to make sure the adaptation worked, since not all changes are automatically adaptable.

Fig. 1 The evolution process from the client's perspective.

This is why recognizing the changes to the specification of a service interface and their impact on client applications is highly desirable and a necessary prerequisite for actually adapting the applications to the new version of the service. Further, assuming that a precise method for service-specification changes existed, it would be extremely useful if one could (a) characterize the changes in terms of their complexity, and (b) semi-automatically develop adapters for migrating clients from older interface versions to newer ones.

In this work, we introduce WSDARWIN, a domain-specific differencing method to compare (a variety of) web-service interfaces. Most frequently, services are developed following two approaches: operation-centric, whose interfaces are specified as Web Service Description Language (WSDL)[1] files, and data-centric (REST), which are specified as Web Application Description Language (WADL)[2] files. Although the two approaches are quite different in the syntax they use to specify web services and their associated technologies, they share a palette of building elements, namely functions and data. WSDARWIN takes advantage of this fundamental commonality to produce accurate comparison results in an efficient and scalable manner for service interfaces regardless of their specification syntax. In this work, we compare

[1] `http://www.w3.org/TR/wsdl`

[2] `http://www.w3.org/Submission/wadl/`

WSDARWIN with our old comparison approach VTRACKER [6] and discuss their differences with respect to performance and scalability. Finally, we apply WSDARWIN on Unicorn[3], W3C's unified validator and Amazon Elastic Cloud Computing (EC2) web service and we present some special cases to demonstrate how the comparison method is used and how its results are presented.

In addition to comparing pairs of specifications to recognize their differences, we are also interested in analyzing the long-term evolution of real world services. We have already presented an empirical study [6], where we analyzed a set of commercial WSDL web services including the Amazon Elastic Cloud Computing (Amazon EC2)[4], the FedEx Package Movement Information and Rate Services[5], the PayPal SOAP API[6] and the Bing search service[7], using VTRACKER, as a comparison method. In that work, we studied the evolution of the aforementioned services and reported our findings on evolution patterns, we identified particular change scenarios and discussed them with respect to their impact on potential client applications and, finally, we correlated these changes with business decisions concerning the services in an effort to reason about the evolution of each service. In this chapter, we extend the findings of this empirical study by providing additional statistics about the changes that the examined services underwent and, more importantly, we provide a classification of the service change scenarios according to their impact on client applications.

The rest of the chapter is organized as follows. In Section 2 we present the extended results of our empirical study on the evolution of WSDL services and we present the classification of service changes. In Section 3, we introduce WSDARWIN as a comparison method for service interfaces and demonstrate its usage on a WSDL and a WADL service. Section 4 provides an overview of the literature related to our work. Finally, Section 5 concludes this chapter and discusses some of our future plans.

2 Study of Web Service Evolution

Before developing methods and tools to support the evolution process of web services, it is important to first study and understand how service interfaces change. This way, we can identify what is important to pay attention to and what can be simplified in order to build improved automated processes. In our work, we have studied five real-world web services offered by companies in the industry of web applications, whose evolution spans across different time periods and exhibits interesting evolution patterns.

[3] http://code.w3.org/unicorn/

[4] http://aws.amazon.com/ec2/

[5] http://www.fedex.com/us/developer

[6] https://www.paypalobjects.com/en_US/ebook/PP_APIReference/architecture.html

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

- **Amazon EC2**. The Amazon Elastic Compute Cloud is a web service that provides resizable compute capacity in the cloud. We studied the history of the web service across 18 versions of its WSDL specification, dating from 6/26/2006 to 8/31/2010.
- The **FedEx Rate Service** operations provide a shipping rate quote for a specific service combination depending on the origin and destination information supplied in the request. We studied 9 versions of this service.
- The **FedEx Package Movement Information Service** operations can be used to check service availability, route and postal codes between an origin and destination. We studied 3 versions of this service.
- The **PayPal SOAP API** Service can be used to make payments, search transactions, refund payments, view transaction information, and other business functions. We studied 4 versions of this service.
- The **Bing Search** service provide programmatic access to Bing content SourceTypes such as Image, InstantAnswer, MobileWeb, News, Phonebook, RelatedSearch, Spell, Translation, Video, and Web. We studied 5 versions of this service.

2.1 Analyzing the evolution of the services

Table 1 shows the evolution profile of all the examined services in terms of data types and operations. Each row corresponds to a service version. Columns 3-8 report the percentage of types and operations in this version that underwent edits (**C**hanges, **D**eletions, **A**dditions) from the previous version. The change columns include two types of changes: renaming or other changes in the "signature" of the object (type or operation), *i.e.,* the attributes of the particular XML element and changes that were propagated from children nodes. For example, if the input or output of an operation or the contained elements of a type are changed, then these changes are propagated to the parent element.

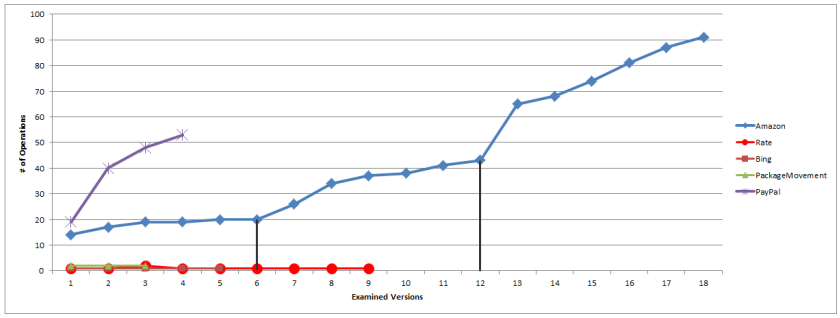
Amazon EC2, as it can be seen from the tables, followed a very distinct pattern of evolution. The developers chose to augment a single service with new operations as they were being developed. For this reason, we observe many additions and changes and a complete lack of deletions. Although this policy eventually produced a rather long WSDL file, it was also prudent in the sense that deleting an operation creates a non-recoverable situation. In such a case a client application should be changed and recompiled. Furthermore, we can observe a correlation between adding new operations and adding new types. This is because in the Amazon services there is a 2-to-1 relationship between types and operations (one input type and one output type for each operation). The changes in the types are usually because of enhancements in previous functionality or to accommodate new functionality. In version 6, we can observe a special case: there are small changes and deletions in types and no other activity. Upon closer examination, it becomes clear that this change represents, in fact, a refactoring.

Table 1 The evolution profile of types and operations in the studied services.

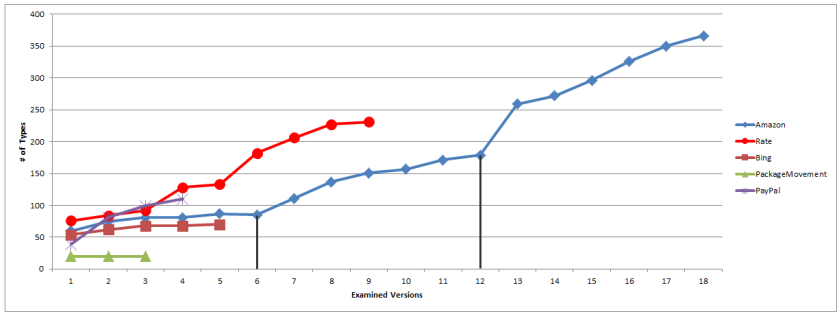
Service	Ver	Types			Operations		
		C(%)	D(%)	A(%)	C(%)	D(%)	A(%)
Amazon EC2	2	5.00	0.00	25.00	0.00	0.00	21.43
Amazon EC2	3	1.33	0.00	8.00	0.00	0.00	11.76
Amazon EC2	4	2.47	0.00	0.00	0.00	0.00	0.00
Amazon EC2	5	7.41	0.00	7.41	0.00	0.00	5.26
Amazon EC2	6	2.30	2.30	0.00	0.00	0.00	0.00
Amazon EC2	7	4.71	0.00	30.59	0.00	0.00	30.00
Amazon EC2	8	0.00	0.00	23.42	0.00	0.00	30.77
Amazon EC2	9	26.28	0.00	10.22	2.94	0.00	8.82
Amazon EC2	10	0.66	0.00	3.97	2.70	0.00	2.70
Amazon EC2	11	0.00	0.00	8.92	0.00	0.00	7.89
Amazon EC2	12	1.17	0.00	4.68	0.00	0.00	4.88
Amazon EC2	13	1.68	0.00	44.69	0.00	0.00	51.16
Amazon EC2	14	1.54	0.00	5.02	0.00	0.00	4.62
Amazon EC2	15	5.88	0.00	8.82	0.00	0.00	8.82
Amazon EC2	16	0.34	0.00	10.14	0.00	0.00	9.46
Amazon EC2	17	1.53	0.00	7.36	0.00	0.00	7.41
Amazon EC2	18	12.00	0.00	4.57	0.00	0.00	4.60
FedEx Rate	2	26.32	1.32	11.84	0.00	0.00	0.00
FedEx Rate	3	14.29	0.00	9.52	0.00	0.00	0.00
FedEx Rate	4	25.00	8.70	47.83	0.00	0.00	100.00
FedEx Rate	5	9.38	0.78	4.69	50.00	50.00	0.00
FedEx Rate	6	10.53	3.01	39.85	0.00	0.00	0.00
FedEx Rate	7	15.38	2.75	15.93	0.00	0.00	0.00
FedEx Rate	8	8.25	0.97	11.17	0.00	0.00	0.00
FedEx Rate	9	18.06	0.44	0.44	0.00	0.00	0.00
Bing	2.1	11.29	0.00	14.81	0.00	0.00	0.00
Bing	2.2	7.35	1.61	11.29	0.00	0.00	0.00
Bing	2.3	2.94	0.00	0.00	0.00	0.00	0.00
Bing	2.4	1.43	0.00	2.94	0.00	0.00	0.00
PayPal	53.0	12.35	0.00	107.69	0.00	0.00	110.53
PayPal	62.0	7.07	0.00	22.22	0.00	0.00	20.00
PayPal	65.1	1.82	0.00	11.11	0.00	0.00	10.42
FedEx Pack.	3	10.00	0.00	0.00	0.00	0.00	0.00
FedEx Pack.	4	5.00	0.00	0.00	0.00	0.00	0.00

The FedEx services (Rate and Package Movement) do not follow the same evolution pattern. These services have a very small number of operations (1 and 2 respectively), which rarely change. On the other hand, the data types evolve vigorously with changes, deletions and additions of new types especially in the Rate service. An interesting change in the Rate service occurred between versions 3 and 4. Until version 3 the service offered a single operation named `getRate`. In version 3, a second operation, named `rateAvailableServices`, was introduced. In version 4, however, the new operation was promptly deleted, `getRate` was renamed to `getRates`, and based on the reorganization of the types, it appears that the responsibilities of the deleted operation were merged into the original one.

Bing and PayPal have both had a relatively short lifecycle but still exhibit interesting differences between them. Bing's history has been relatively stable, with few modifications given also the small number of elements in its WSDL specification (1 operation and between 54 and 70 types). PayPal, on the other hand, follows an expansion pattern similar to the one Amazon follows; it is consistently enhanced with new operations. The great increase observed in Figure 2(a) in the number of operations between the first two examined versions of PayPal is because there are a lot of intermediate versions for which we have no data.



(a) Evolution of number of operations



(b) Evolution of number of types

Fig. 2 The evolution of the examined services

Figures 2(a) and 2(b) show the evolution of the operations and types of the examined services. An interesting observation from these figures concerns the Amazon service, where we can see that the particular service seems to have three distinct phases: the first is from version 1 to version 6, the second is from 7 to 12 and the third from 13 to 18. These phases are the result of the business decisions that have been described in [6].

2.2 Classification of Service Changes

Based on the discussion about specific changes that happened in the web services we examined in [6], we propose a classification of these changes based on their impact on client applications. Because of the distributed nature of service systems, clients usually have very little information to understand the changes in web services and contemplate their impact on their applications. Therefore, accurately recognizing and characterizing service changes will facilitate clients reason about these changes and systematically build adapters for their applications. We distinguish three types of changes with respect to their impact on clients.

1. *No-effect* changes do not impact the client at all. The client functionality is not disrupted and neither is the interface, which practically means that the client can still operate using the old stub. Changes in this category include adding new types (as long as these types are not used by existing operations) and adding new operations (assuming that the semantics of the service are preserved and there are no interdependencies between the new and the old operations).
2. *Adaptable* changes affect the interface of the client, but the functionality of the service remains the same. These changes, from the point of view of the provider, usually correspond to refactorings on the source code of the service. In other words, they are changes meant to improve the design of the service and leave the functionality unaffected. They can be easily addressed by generating a new stub and changing the old stub, still used by the client application, to invoke the new one and thus the evolved service [7]. This way we avoid changing the client code by modifying only autogenerated code. Changes in this category include refactorings, renaming and changing input or output for an operation (assuming that the new input or output are existing types and not new ones).
3. *Non-recoverable* changes imply that the functionality of the service is affected, in a way that the client breaks and we cannot address the issue without changing and recompiling the client code. In some cases, the change is so subtle as not to affect the interface of the client. In other words, the client still works but the results produced are not the desired ones. The problem in this case can be identified by means of unit and regression testing. Removing elements from the service interface (without replacing them) is a non-recoverable change.

Even after the identification of detailed changes between versions of the service interface and the classification of these changes, the adaptation of client applications may still not be plausible. Even in the first two categories, functionality may be affected and this impact may seem invisible or easily addressable by examining just the service interface. For this reason, testing of the adapted client application may still be needed and additional (manual) effort may be required.

2.3 Implications of the Empirical Study

Apart from drawing conclusions for the evolution of web service interfaces, including evolution patterns, lifecycles, good and bad practices, through the empirical studies we identified types of simple or more complex, but definitely recurring, changes. These examples, along with ones drawn from our experience in designing and developing software systems, have been used to design the comparison component of WSDARWIN. The study has shown us what kind of changes to expect and the instances of these changes in commercial web services have helped us to understand how we can automatically identify such changes.

On the other hand, the classification of service changes primarily contributes to the adaptation and generally the evolution of client applications. In a recent work [7], we propose an adaptation algorithm that automatically adapts client applications to adaptable changes of the service interface. The knowledge of what category the change belongs to, can help us identify whether automatic adaptation can be applied. The classification can also improve the comparison method. For instance, in case of refactorings, these types of changes have very specific mechanics (see the work by Fowler [8]), which can be translated to comparison rules in WSDARWIN, thus expanding the system's capabilities to identify a greater variety of changes.

3 WSDARWIN

In order to be able to systematically adapt client applications to the changes of the web services on which they rely, we, first, should be able to accurately recognize the changes a web service undergoes. In developing a web-service differencing algorithm, one should consider two quality properties: (a) the efficiency and scalability of the algorithmic process, and (b) the understandability of the output it produces. The process has to be efficient and scalable because service-interface descriptions can be quite lengthy and complex, as they may contain many and complex types and numerous operations. On the other hand, as the differencing process is usually preformed in service of another task, such as adaptation for example, its output has to be understandable by the developers and it also has to be designed to be easily consumed by any downstream automated process.

In the WSDARWIN comparison method, we ensure efficiency by using a concise, domain-specific model to represent the relevant information of a service interface. The model captures the most important information of a service's elements such as *names*, *types*, their *structure* and the *relationships* with each other, thus, providing a simpler, more lightweight syntactic representation of the service representation than either WSDL or WADL. In addition, the algorithm employs certain heuristics on name comparisons to further improve the efficiency. The rationale underlying these heuristics is that within the same service (even between versions) names are unique and can therefore be treated as IDs. The use of the same name for different elements

is not likely (and in many cases it is not allowed). For this reason, it only makes sense to compare strings using exact matching and not partial matching techniques such string-edit distance. Furthermore, instead of comparing named XML nodes like VTRACKER, WSDARWIN compares model entities based on their specific type (e.g. operations with operations, complexTypes with complexTypes etc.). This way it is not necessary to compare all elements against each other, thus avoiding false results due to fuzzy mapping and gaining further efficiency improvement over VTRACKER.

WSDARWIN's output follows the model shown in Figure 3[8]. Figure 3(a) shows the model used to represent WSDL service interfaces. The operations, which are the invocation points between the provided service and the client application, have input and output types. The type hierarchy is in accordance with the XML Schema specification[9]: `PrimitiveTypes` include strings, integers, boolean etc.; `SimpleTypes` are based on certain restrictions on their values (e.g. enumerations); `ComplexTypes` are composed of other types. The model omits elements that add no further structural information for the clients, such as `messages` and high level `elements` from the schema, which only serve as references. Therefore, only the elements to which these references point were eventually included in the model.

Figure 3(b) corresponds to the WADL interface model. The element `resources` contains a set of `resource` elements, which in turn contain `methods` and these have `requests` and `responses`. Requests consist of a set of `parameters` and the responses, which are usually returned as a file of structured data such as XML or JSON, refer to elements in an XML schema file. The IType hierarchy is the same as in the WSDL model.

In both these models, the containment relationships (denoted by the black diamonds) indicate parent-child relationships between element types. For example, an operation in WSDL has two children: an input type and an output type. The children elements together represent the *structure* of a WS element. Structural information can be used to uniquely identify elements. If two elements across two web-service specifications have the same children, then there is high confidence that they are one and the same element.

Figures 4(a) and 4(b)[10] show examples of the instantiation of the WSDL and WADL models for the Amazon EC2 and the Unicorn validator, respectively. The figures clearly demonstrate the structure of elements, implemented by the parent-children relationship between WS elements as defined in the interface models.

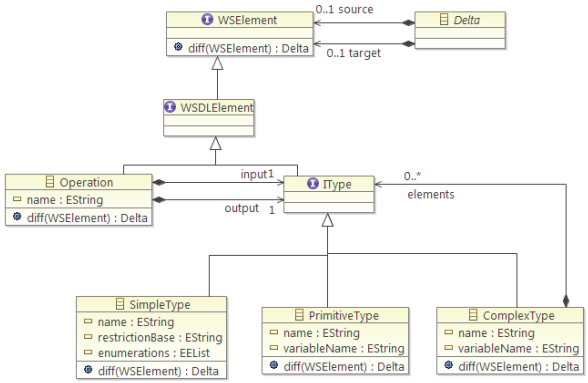
Figure 3(c) models the changes. We can have different types of deltas including *changes*, *additions*, *deletions*, *moves* and *moves and changes*. The two hierarchies are connected through the Bridge design pattern [9] and their relationship is that each delta has a source WS element and a target WS element.

The interface models define the structure and the vocabulary of the diff scripts produced by WSDARWIN; the Delta model defines the annotations for each map-

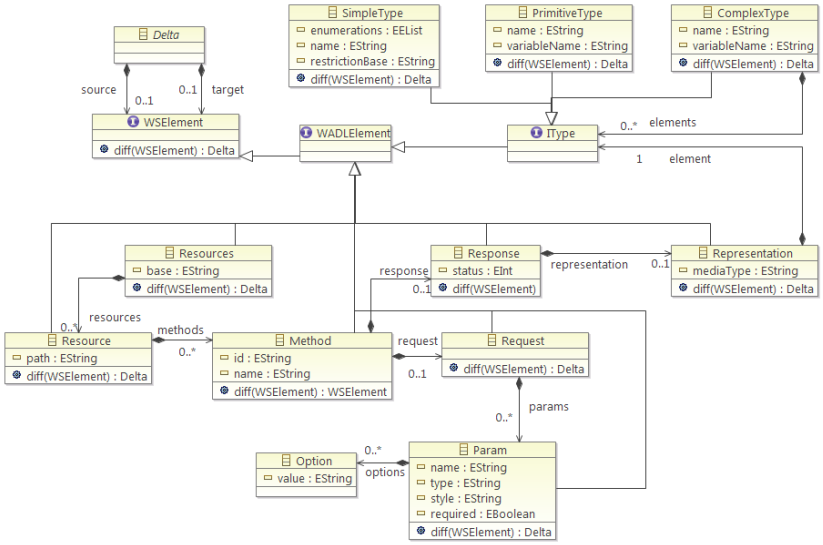
[8] The diagrams were designed using the Eclipse EMF toolkit.

[9] http://www.w3.org/XML/Schema

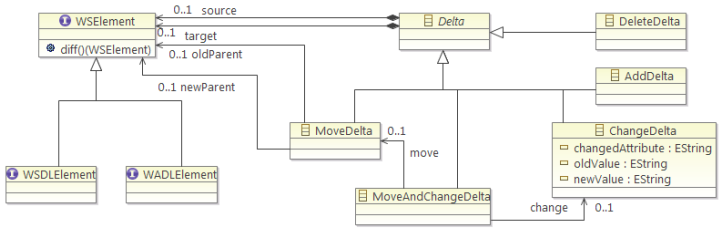
[10] The figures were generated by the Eclipse EMF toolkit.



(a) The WSDL service interface model.

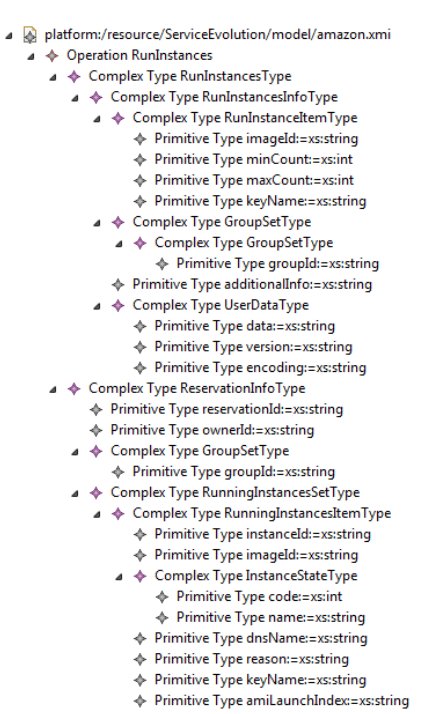


(b) The WADL service interface model.



(c) The WSDarwin delta model.

Fig. 3 The WSDarwin comparison framework.



(a) Amazon EC2 (b) W3C Unicorn CSS validator

Fig. 4 Snippets of WSDL and WADL instances of the WSDARWIN interface models.

ping reported in these scripts. Designing WSDARWIN in this manner, we have striven for a balance of specificity to the syntax of the compared specification (WSDL vs. WADL) and generality in the definition of the changes the interfaces go through. This design, we believe, makes the output clear to web-service system developers and enables them to understand and better reason about the changes in the services. Furthermore, the output is designed with consideration to a downstream automated adaptation process, since it provides a full mapping between the elements and the type of every change so that the process can assess its impact on the client application.

Let us now review WSDARWIN in some detail. For each version v to be examined, WSDARWIN extracts E_v, the set of elements in the specification of the v version of the service. This set contains tuples (id,t,a,s) where id is the identifying attribute of the element (usually the name), t is the type of the element, a is the set of its attributes and s is the structure of the element.

In the context of the WSDARWIN comparison, the ID or the structure can uniquely identify an element. Therefore, if two elements, belonging in two different versions, share at least one of these two properties (ID and structure), then WSDARWIN considers them to be two versions of the same element. Since web service interfaces are artifacts generated by source code, they also follow the programming conventions of the underlying programming language. In principle, two entities in the same file cannot have the same name, or a compilation error occurs. Therefore, we can safely assume that the name of an entity is its unique identifier. On the other hand, we also consider structure to be a unique identifier so as to be able to identify cases of renaming. In the rare case, where the new version contains two elements, one with the same name as the old entity and the other with the old entity's structure but different name, WSDARWIN might get confused, but the diff script exactly the same set of edit operation: one addition and one change.

Note that the set E_v contains elements of all types across the WSDL and WADL specification syntaxes. For every element in a specific version of the web-service $e \in E_v$, WSDARWIN identifies

- A_e: The set of attributes, other than the ID, and
- S_e: The structure of the element, if it is a complex element. Note that, as we mentioned above, the structure refers to the children of complex elements such as input and output types for operations and elements for complex types.

Finally, for each comparison Δ, between two versions $v_1, v_2 \in V$, where V is the set of versions of a web-service specification to be analyzed, we determine the added and deleted matched elements by using the symbols "+" and "-"respectively. Therefore, E_Δ^+ is the set of elements that were added. We also use the symbol "#" to denote mapped elements, e.g. $E_\Delta^\#$.

WSDARWIN relies on a set of rules to map and differentiate the elements between different versions of the service interfaces. Table 2 summarizes the rules we use to compare service interfaces.

Table 2 The definition of rules used by WSDarwin for the comparison of web service interfaces.

	Name of comparison rule	**Rule**
1	Exact matching	$\forall a_{e_1} \in A_{e_1}, \forall a_{e_2} \in A_{e_2} : a_{e_1}.literal = a_{e_2}.literal$
2	Mapping	$\exists e_1, e_2 \in E_\Delta^\# : e_1.t = e_2.t$ and $(e_1.id = e_2.id$ or $e_1.s = e_2.s)$
3	Changed	$\exists (id_i, t_i, a_i, s_i) \in E_\Delta^\#$ and $\exists (id_j, t_j, a'_j, s_j) \in E_\Delta^\#$
4	Propagated change	$\exists (id_i, t_i, a_i, s_i) \in E_\Delta^\#$ and $\exists (id_j, t_j, a_j, s'_j) \in E_\Delta^\#$
5	Matched	$\exists (id_i, t_i, a_i, s_i) \in E_\Delta^\#$ and $\exists (id_j, t_j, a_j, s_j) \in E_\Delta^\#$
6	Added	$\exists e_{v_2} \notin E_\Delta^\#$
7	Deleted	$\exists e_{v_1} \notin E_\Delta^\#$
8	Changed (Renamed)	$\exists (id_i, t_i, a_i, s_i) \in E_\Delta^-$ and $\exists (id'_j, t_j, a_j, s_j) \in E_\Delta^+$
9	Moved	$\exists (id_i, t_i, a_i, s_i) \in E_\Delta^-$ and $\exists (id_j, t_j, a_j, s_j) \in E_\Delta^+$
10	Moved and Changed	$\exists (id_i, t_i, a_i, s_i) \in E_\Delta^-$ and $\exists (id'_j, t_j, a'_j, s'_j) \in E_\Delta^+$

1. The first rule is the exact matching rule. In case of simple attributes (such as the element's ID and attributes belonging in the A_e set of the element), two attribute values are the same if and only if they have the same literal. In case of structure (*i.e.,* the set of children of an element), two elements are considered structurally equal if and only if all their children are equal. Children equality is determined in an iterative manner.
2. The second rule states that two elements are "mapped" to each other, *i.e.,* they are considered the same element across the two interface versions, if their type and at least one of their identifying properties, *i.e.,* ID and structure, match. It is important to note that two mapped elements are not necessarily matched. There can still exist differences in which case a Change Delta is reported. On the other hand, matched elements are always mapped.
3. An element is considered "changed" if its ID was found in both versions but some of the values of its other attributes differ across the two versions.
4. If there is a change in the structure of the element (*i.e.,* its children have changed), the element itself is considered "changed" even if none of the attributes of the parent element have changed. This is because the adaptation process starts from the root element of a service request which is considered to be the operation. Therefore, if some part of its input or its output is affected the operation is still considered affected.
5. If two elements are mapped and no differences are identified, they are labeled as "matched". The need to retain matched elements in the final comparison script is because an automated adaptation process needs a full mapping between the two versions.
6. An "addition" is identified if an element's name (its ID) that did not exist in the old version, but it was not found in the new version.
7. Correspondingly, a "deletion" is identified if an element's name existed in the old version, but it was not found in the new version.
8. In a second phase, the additions and deletions are reexamined to recognize potential changes in the element IDs or moves. If an element is identified as deleted from the old version and another element as added in the new version and the two elements have identical structure but differ with respect to their IDs then these elements are labeled as "changed (renamed)".
9. In a similar scenario, where elements are mapped between the deleted and added sets, these elements are marked as "moved". The reason they couldn't be identified in the first run of the comparison is because the process follows the structure of the service interface and elements are compared only in the context of their parents. Legitimate moves in a WSDL interface include primitive types being moved between complex types. Another also legal, but less probable, move can occur when two operations exchange their input or output types. In WADL, where the structure is more complicated, we can have `resource` elements being moved between `resources` elements and methods being moved between `resource` elements. Moves involving data types are also possible in this syntax.

10. If the moved elements also differ in their structures or their IDs, they are labeled as "moved and changed". If they differ with respect to both structure and ID, then they are considered different elements and are report as an addition and a deletion.

Algorithm 1 diff(e_1, e_2) WSDARWIN service interface comparator

```
1: Compare the attributes of the two elements.
2: if Changes are detected then
3:     Set ElementDelta to ChangeDelta(e1,e2)
4: end if
5: for all c1 ∈ Children(e1) do
6:     for all c2 ∈ Children(e2) do
7:         if ¬Mapped(c1,c2) then
8:             Add DeleteDelta(c1) OR AddDelta(c2) to ElementDelta
9:             for all cc1 ∈ Children(c1) do
10:                Add DeleteDelta(cc1) to DeleteDelta(c1)
11:            end for
12:            for all cc2 ∈ Children(c2) do
13:                Add AddDelta(cc2) to AddDelta(c2)
14:            end for
15:        else
16:            Call diff(c1,c2)
17:            Add result to ElementDelta
18:            if The result contains only MatchDeltas AND ElementDelta != null
               then
19:                Set ElementDelta to MatchDelta(e1,e2)
20:            else
21:                //Change propagated.
22:                Augment ElementDelta with ChangeDelta(e1,e2)
23:            end if
24:        end if
25:    end for
26: end for
```

Algorithm 2 findMoveDeltas$(Delta)$

```
1: for all AddDelta(e2) AND DeleteDelta(e1) ∈ Delta do
2:     if Mapped(e1,e2) then
3:         if Changed(e1,e2) then
4:             Create MoveAndChangeDelta(e1,e2)
5:             Replace DeleteDelta(e1) with MoveAndChangeDelta(e1,e2)
6:         else
7:             Create MoveDelta(e1,e2)
8:             Replace DeleteDelta(e1) with MoveDelta(e1,e2)
9:         end if
10:    end if
11: end for
```

Based on the model and using the rules, in the first phase, the differencing method performs pairwise comparisons between the elements of the service interfaces starting from the more complex ones, such as the WSDL or WADL files themselves, and going down the hierarchy of the service elements as shown by Algorithm 1. First, the algorithm reports any changes in the attributes of the element (using the 3rd rule) or in the ID of the element (using the 8th rule) *(steps 1-4)*. Second, the children of the compared elements are mapped according to the 2nd rule *(step 7)*. Those that were not mapped are considered added, according to the 6th rule, or deleted, according to the 7th rule *(step 8)*. If a complex element is added or deleted all of its children are also added or deleted to acquire a full mapping between the two versions *(steps 9-14)*. The elements that were mapped are then compared *(step 16)*. The comparisons continue this way until they reach simple elements, such as XSD elements or WADL `param` elements, which are only compared based on their attributes since they have no children and the comparison result is returned to the parent. In the final step, the algorithm checks if the children of the compared elements and the children of their children are matched according to the 5th rule, then the compared elements are matched as well *(step 19)*. Otherwise, a change is propagated to the parent according to the 4th rule *(step 22)*. In a second phase shown by Algorithm 2, WSDARWIN tries to identify moved elements among the added and the deleted ones. In the first phase, additions and deletions are identified within the scope of an element. In the second phase, the hierarchy is collapsed and additions and deletions are reexamined to detect moves based on the 9th and the 10th rule.

3.1 WSDARWIN *versus* VTRACKER

VTRACKER, the first method we used for web-service differencing, is a generic domain-agnostic differencing algorithm that can be used to compare heterogeneous interfaces, *i.e.,* interfaces described in different schemas. In other words, VTRACKER can be used to compare any pair of XML documents. For this reason, this method uses fuzzy mapping and partial matching. For the former option, since we don't always know a mapping between the elements of the two interfaces, the algorithm compares all elements with each other (regardless of their type) and establishes a mapping based on their structural similarity. As far as the partial matching is concerned, the algorithm uses the notion of distance to compare elements with each other. Then, using a stable marriage algorithm it matches the elements with the lowest edit distance. VTRACKER can be configured to include information about the specific XML syntax used by the files to be compared. In our previous study [6], we configured VTRACKER to work with WSDL interfaces. In the end, the output produced by the algorithm is a text-like document containing the appropriate XML edit operations to go from the first file to the second.

WSDARWIN, on the other hand, is a comparison method tailored to the web-service domain and it is developed from the beginning with knowledge about the structure of the interfaces, thus improving on quality properties such scalability and

understandability. Fuzzy mapping can cause problems in the case of elements of different types named in a similar manner if they correspond to the same concept. In the case of web services, the convention is to name operations and their input and output types similarly to denote their relationship. Fuzzy mapping and partial matching also contribute to decreased efficiency and accuracy: when the algorithm considers a variety of increasingly relaxed methods for establishing correspondence between two elements, then it has to perform more computations (resulting to inefficiency) and it risks establishing correspondence on more "risky" grounds (resulting to inaccuracy). WSDARWIN takes advantage of the fact that web services share a common palette of elements, regardless of their syntax, namely data and functionality. In other words, this method is domain-specific, but technology-agnostic. Furthermore, having a priori knowledge, it compares elements according to their types and taking advantage of naming conventions, it uses exact matching to compare literals. Finally, the output of WSDARWIN is based on the Deltas and follows the structure of the service interface, which makes it not only understandable but also easily consumable by automated adaptation techniques. Table 3 summarizes the comparison between VTRACKER and WSDARWIN.

Table 3 Comparison between VTRACKER and WSDARWIN.

VTRACKER	WSDARWIN
Domain-agnostic	Domain-specific
Technology-specific	Technology-agnostic
Heterogeneous comparisons - Can be applied on any XML-like file	Homogeneous comparisons - Can be applied only on the WS domain
Less efficient - Fuzzy mapping - Partial matching	More efficient - Mapping according to type, structure and identifier - Exact matching (same literal)
Free text output - XML edit operations	Structured output - Deltas - Directly consumable by CASE tools

Figure 5 shows the execution time of VTRACKER and WSDARWIN with respect to the size of the compared service interfaces. Time measurements were performed in a machine with an Inter Core 2 Duo 1.87 GHz CPU, 3 GB RAM and 64-bit operating system. This figure clearly demonstrates the scalability of WSDARWIN even in the presence of large services. VTRACKER approximates an exponential execution time while WSDARWIN's is linear. Apart from the fuzzy mapping and partial matching, another factor that contributes to VTRACKER's large execution time is the fact that when comparing the structure of an element, the method has to resolve and compare references and this resolution takes place for each reference. WSDARWIN, on the other hand, resolves references only once during the parsing of the service interface and replaces the references with containment relationship, so the method avoids the time to seek for the element corresponding to a reference every time it encounters one.

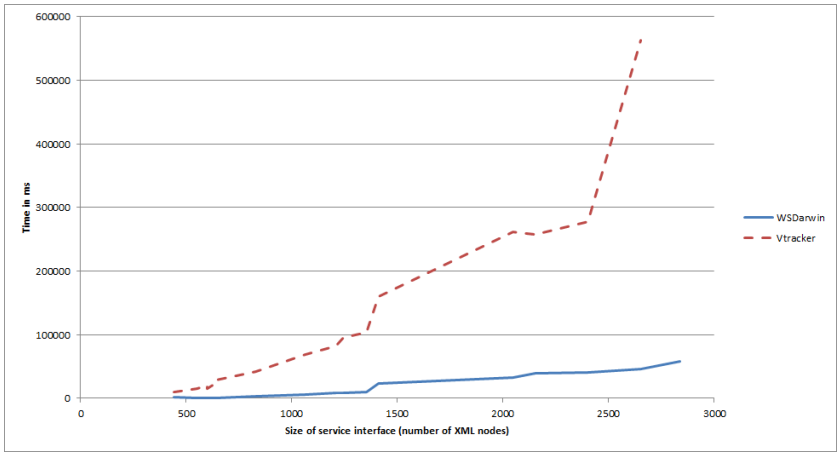


Fig. 5 Comparison between WSDARWIN and VTRACKER in terms of their execution time.

3.2 Applying WSDARWIN *on the Comparison of Service Interfaces*

In this section, we demonstrate with examples how the WSDARWIN differencing method can be used to compare different versions of service interfaces. We applied the method on Amazon EC2, which has a WSDL-based interface, and Unicorn, which has a WADL-based interface. We chose these examples to show that given the proper model to represent the service interface, the comparison method, which is based on the delta model, can be applied to compare the interfaces regardless of their underlying specification technology.

Figure 6 shows a snippet of the output of WSDARWIN for the Amazon EC2 service. The diff script follows the hierarchy of the WSDL interface starting with the operations and then their input and output types. Each line is prefixed with the type of the edit operation performed for each element. The detection of move operations is activated for the script in Figure 6(b), and deactivated for the script reported in Figure 6(a). Comparing the two figures, we observe that the move operations are first perceived as additions and deletions, in the first phase of the comparison algorithm. In the second phase, the deletions are replaced by move operations but the additions are kept in the diff script.

In this example, we have a case of an "Inline Type" refactoring as described in our previous work [6]. As it can be seen from the figure, such a refactoring occurs when a type (`RunInstancesInfoType`), which is nested into another complex type (`RunInstancesType`, is deleted from the service and its constituent elements are *all* added in the parent type. By identifying the edit operations as moves and not as actual deletions, we can characterize this change as *adaptable* according to our classification. This is because the data exists in both versions but is "packaged" differently.

```
1. ChangeOperation  RunInstances -> RunInstances
2.    Change  ComplexType:RunInstancesType -> :RunInstancesType
3.        Add PrimitiveType       -> instanceType:string
4.        Add PrimitiveType       -> imageId:string
5.        Add PrimitiveType       -> keyName:string
6.        Add PrimitiveType       -> minCount:int
7.        Add PrimitiveType       -> maxCount:int
8.        Delete  ComplexTypeinstancesSet:RunInstancesInfoType ->
9.            Delete  PrimitiveType  keyName:string ->
10.           Delete  PrimitiveType  imageId:string ->
11.           Delete  PrimitiveType  minCount:int ->
12.           Delete  PrimitiveType  maxCount:int ->
13.       Match   PrimitiveType  addressingType:string -> addressingType:string
14.       Match   ComplexTypegroupSet:GroupSetType -> groupSet:GroupSetType
15.           Match   ComplexTypeitem:GroupItemType -> item:GroupItemType
16.               Match   PrimitiveType  groupId:string -> groupId:string
17.       Match   ComplexTypeuserData:UserDataType -> userData:UserDataType
18.           Match   PrimitiveType  data:string -> data:string
19.       Match   PrimitiveType  additionalInfo:string -> additionalInfo:string
```

(a) Diff script without the detection of move operations

```
1. Change Operation   RunInstances -> RunInstances
2.    Change  ComplexType :RunInstancesType -> :RunInstancesType
3.        Add PrimitiveType        -> instanceType:string
4.        Add PrimitiveType        -> imageId:string
5.        Add PrimitiveType        -> keyName:string
6.        Add PrimitiveType        -> minCount:int
7.        Add PrimitiveType        -> maxCount:int
8.        Delete  ComplexType instancesSet:RunInstancesInfoType ->
9.            Move    PrimitiveType   keyName:string
                      instancesSet:RunInstancesInfoType ->:RunInstancesType
10.           Move    PrimitiveType   imageId:string
                      instancesSet:RunInstancesInfoType ->:RunInstancesType
11.           Move    PrimitiveType   minCount:int
                      instancesSet:RunInstancesInfoType ->:RunInstancesType
12.           Move    PrimitiveType   maxCount:int
                      instancesSet:RunInstancesInfoType ->:RunInstancesType
13.       Match   PrimitiveType   addressingType:string -> addressingType:string
14.       Match   ComplexType groupSet:GroupSetType -> groupSet:GroupSetType
15.           Match   ComplexType item:GroupItemType -> item:GroupItemType
16.               Match   PrimitiveType   groupId:string -> groupId:string
17.       Match   ComplexType userData:UserDataType -> userData:UserDataType
18.           Match   PrimitiveType   data:string -> data:string
19.       Match   PrimitiveType   additionalInfo:string -> additionalInfo:string
```

(b) Diff script with the detection of move operations

Fig. 6 Snippet of the diff script between two versions of the Amazon EC2 service.

Also, edit operations of children elements are propagated as changes to the parent element. This is so that the adaptation process knows as early as possible which are the operations that are affected, since these are the contact elements between the service interface and client applications. For example, as it can be seen in the figure, because of the changes (additions and deletions) in the input of the `RunInstances` operation, these changes affect the operation which is marked as changed, despite not being directly changed.

Figure 7 shows the output of WSDARWIN for the CSS validator service of Unicorn. The only major difference between the Unicorn and the Amazon diff scripts is that the former follows the WADL hierarchy. The edit operations are reported in exactly the same manner based on the delta model. In this case, we also have an instance of an attribute change (line 13). These changes are reported by identifying which attribute was changed (in this case attribute “name” of method “CssValida-

```
1. Change  WADLFile files/unicorn/css-validator/css-validatorV1.wadl
2.                  -> files/unicorn/css-validator/css-validatorV8.wadl
3.     Change   Resources    http://jigsaw.w3.org/css-validator/
                        -> http://qa-dev.w3.org:8001/css-validator/
                        @base  http://jigsaw.w3.org/css-validator/
                            -> http://qa-dev.w3.org:8001/css-validator/
4.         Change   Resource validator -> validator
5.             Match    Method   CssValidationUri (GET) -> CssValidationUri (GET)
6.                 Match    Request  Request -> Request
7.                     Match    Param    usermedium -> usermedium
8.                     Match    Param    output -> output
9.                     Match    Param    uri -> uri
10.                    Match    Param    lang -> lang
11.                    Match    Param    warning -> warning
12.                    Match    Param    profile -> profile
13.            Change   Method   CssValidationText (POST) -> CssValidationText (GET)
                                @name    POST -> GET
14.                Match    Request  Request -> Request
15.                    Match    Param    text -> text
16.                    Match    Param    usermedium -> usermedium
17.                    Match    Param    output -> output
18.                    Match    Param    lang -> lang
19.                    Match    Param    warning -> warning
20.                    Match    Param    profile -> profile
21.            Match    Method   CssValidationFile (POST) -> CssValidationFile (POST)
22.                Match    Request  Request -> Request
23.                    Match    Param    file -> file
24.                    Match    Param    usermedium -> usermedium
25.                    Match    Param    output -> output
26.                    Match    Param    lang -> lang
```

Fig. 7 The diff script between two versions of the WADL-based CSS validator of Unicorn.

tionText") prefixed by the symbol "@" for attribute, along with its old value and its new value. An attribute change subsumes a propagated change, since both edit operations mark the element as affected. For this reason, we do not need an additional type delta for either edit operation.

As we have already mentioned, while the structure and the vocabulary of the diff script are dictated by the underlying syntax model, the Deltas are used as annotations. This demonstrates and emphasizes the fact that WSDARWIN is technology-agnostic; regardless of the syntax model, the Delta language can be applied to provide the comparison context of the diff script.

4 Related Work

Our work relates to differencing, WSDARWIN's contribution, and service evolution, the substance of our empirical study.

4.1 Model- and Tree-Differencing Techniques

Fluri *et al.* [5] proposed a tree-differencing algorithm for fine-grained source code change extraction. Their algorithm takes as input two abstract syntax trees and extracts the changes by finding a match between the nodes of the compared trees.

Moreover, it produces a minimum edit script that can transform one tree into the other given the computed matching. The proposed algorithm uses the bi-gram string similarity to match source code statements (such as method invocations, condition statements, and so forth) and the sub-tree similarity of Chawathe et al. [3] to match source code structures (such as if statements or loops). The method also uses names and types as IDs to map elements and can identify primarily changes, additions, deletions and moves for different types of elements.

Kelter et al. [10] proposed a generic algorithm for computing differences between UML models encoded as XMI files. The algorithm first tries to detect matches in a bottom-up phase by initially comparing the leaf elements and subsequently their parents in a recursive manner until a match is detected at some level. When detecting such a match, the algorithm switches into a top-down phase that propagates the last match to all child elements of the matched elements in order to deduce their differences. The algorithm reports four different types of differences, namely structural (denoting the insertion or deletion of elements), attribute (denoting elements that differ in their attributes' values), reference (denoting elements whose references are different in the two models) and move (denoting the move of an element to another parent element). Although the method does not use IDs to map elements, they are necessary to identify moves. For this reason, custom ones are constructed using the name of the element and its path along the XMI tree.

Xing and Stroulia [15] proposed the *UMLDiff* algorithm for automatically detecting structural changes between the designs of subsequent versions of object-oriented software. The algorithm produces as output a tree of structural changes that reports the differences between the two design versions in terms of additions, removals, moves, renamings of packages, classes, interfaces, fields and methods, changes to their attributes, and changes of the dependencies among these entities. UMLDiff employs two heuristics (*i.e.,* name-similarity and structure-similarity) for recognizing the conceptually same entities in the two compared system versions. These two heuristics enable UMLDiff to recognize that two entities are the same even after they have been renamed and/or moved. The UMLDiff algorithm has been employed for detecting refactorings performed during the evolution of object-oriented software systems, based on UMLDiff change-facts queries [16].

Recently, Xing [14] proposed a general framework for model comparison, named *GenericDiff*. While it is domain independent, it is aware of domain-specific model properties and syntax by separating the specification of domain-specific inputs from the generic graph matching process and by making use of two data structures (*i.e.,* typed attributed graph and pair-up graph) to encode the domain-specific properties and syntax so that they can be uniformly exploited in the generic matching process. Unlike the aforementioned approaches that examine only immediate common neighbors, GenericDiff employs a random walk on the pair-up graph to spread the correspondence value (*i.e.,* a measurement of the quality of the match it represents) in the graph.

In our previous work [6], we adopted VTRACKER to recognize the differences between two versions of a web-service interface. VTRACKER is designed to com-

pare and recognize the similarities and differences between XML documents, based on the Zhang-Shasha tree-edit distance [17] algorithm.

WSDARWIN is tailored around a very specific domain, that of web services. Therefore, a lot of domain-specific information and characteristics are imbued in the comparison method. However, we do borrow some fundamental differencing techniques from the works described in this section. For example, many methods employ the concept of a model to describe the compared artifacts. In fact, the underlying model is the one that will determine the accuracy and the efficiency of the comparison method. Second, the use of identifiers for mapping compared elements is a widely used technique, also present in the VTRACKER algorithm. Finally, the propagation of changes as described in WSDARWIN, is a similar technique as the top-down/bottom-up approach used by Kelter et al.

Table 4 Comparison between differencing techniques.

Method	Type	Edit Operations	IDs	Exact Matching	Model
Kelter	generic	CAD(M)	No(Yes)	No	UML/XMI
Fluri	domain-specific	CADM	Yes	No	AST
UMLDiff	domain-specific	CADMX	Yes	No	Custom/UML
GenericDiff	generic	CADM	Yes	No	UML
VTRACKER	generic	CADM	Yes	No	XML
WSDARWIN	domain-specific	CADM	Yes	Yes	Custom/WS

Table 4 positions WSDARWIN among the aforementioned works with respect to whether they are generic or domain-specific, what kind of edit operations they can identify (**C**hange, **A**ddition, **D**eletion, **M**ove, comple**X** changes), if they employ IDs for the mapping of elements, whether they use exact matching in the comparison and finally what is the underlying model.

4.2 Service-Evolution Analysis

In addition to web-service (and web-service version) comparison, substantial efforts have been dedicated to the task of web-service evolution analysis. Wang and Capretz [13] proposed an impact-analysis model as a means to analyze the evolution of dependencies among services. By constructing the intra-service relation matrix for each service (capturing the relations among the elements of a single service) and the inter-service relation matrix for each pair of services (capturing the relations among the elements of two different services) it is possible to calculate the impact effect caused by a change in a given service element. A relation exists from element x to element y if the output elements of x are the input elements of y, or if there is a semantic mapping or correspondence built between elements of x and y. Finally, the intra- and inter-service relation matrices can be employed to support

service change operations, such as the addition, deletion, modification, merging and splitting of elements.

Aversano et al. [2] proposed an approach, based on Formal Concept Analysis, to understand how relationships between sets of services change across service evolution. To this end, their approach builds a lattice upon a context obtained from service description or operation parameters, which helps to understand similarities between services, inheritance relationships, and to identify common features. As the service evolves (and thus relationships between services change) its position in the lattice will change, thus highlighting which are the new service features, and how the relationships with other services have been changed. This approach is useful to study the evolution of similar interchangeable services.

Ryu et al. [12] proposed a methodology for addressing the dynamic protocol evolution problem, which is related with the migration of ongoing instances (conversations) of a service from an older business protocol to a new one. To this end, they developed a method that performs change impact analysis on ongoing instances, based on protocol models, and classifies the active instances as migrateable or non-migrateable. This automatic classification plays an important role in supporting flexibility in service-oriented architectures, where there are large numbers of interacting services, and it is required to dynamically adapt to the new requirements and opportunities proposed over time.

In a similar vein, the WRABBIT project [4] proposed a middleware for wrapping web services with agents capable of communication and reflective process execution. Through their reflective process execution, these agents recognize run-time "conversation" errors, *i.e.,* errors that occur due to changes in the rules of how the partner process should be composed and resolve such conversation failures.

Pasquale et al. [11] propose a configuration management method to control dependencies between and changes of service artifacts including web services, application servers, file systems and data repositories across different domains. Along with the service artifacts, Smart Configuration Items (SCIs), which are in XML format, are also published. The SCIs have special properties for each artifact such as host name, id etc. Interested parties (like other application servers) can register to the SCIs and receive notifications for changes to the respective artifact by means of ATOM feeds and REST calls. Using a discovery mechanism the method is able to identify new, removed or modified SCIs. If a SCI is identified as modified, then the discovery mechanism tracks the differences between the two items and adds them as entries in the new SCI. The changes that can be identified are delete, add, modify a property or delete, add, modify a dependency.

Andrikopoulos et al. [1] propose a service evolution management framework. The framework generally aims to support service providers evolve their services. It contains an abstract technology-agnostic model to describe a service system in its entirety, specifying all artifacts such as service interfaces, policies, compositions etc. and divide the artifacts in public and private. This division implies that the management framework has knowledge about the service's back-end functionality, which in turn means that it can be used only by the provider. The authors also propose a classification for the changes based on the basic operations (additions, deletions

etc.) and guidelines on how to evolve, validate and conform service specifications to older versions. Although such a management framework may lead to a smooth evolution process, inconsistencies may still occur between services and their clients. Therefore, support to clients is equally important.

Table 5 Comparison between service evolution works.

Method	Dependencies	Client Support	Level
Wang	Inter	Yes	Protocol
Aversano	Inter	No	Interface
WRABBIT	Inter	Yes	Protocol
Pasquale	Intra	Yes	Interface
Ryu	Inter	No	Protocol
Andrikopoulos	Intra	No	Source Code
WSDARWIN	Intra	Yes	Interface

Table 5 summarizes the comparison between WSDARWIN and these other projects along 3 dimensions:

- what kind of dependencies the method examines:
  - inter-dependencies, requiring knowledge about different parts of the service system;
  - intra-dependencies, focusing on a particular part;
- whether the method provides any support to consumers of the service.
- what is the architectural level the method uses to study the service systems:
  - business protocol level, where the method needs information about various services in the system;
  - interface, where the method only examines boundary artifacts, such as service interfaces;
  - source code, where the method needs back-end information.

5 Conclusion and Future Work

In this chapter, we introduced WSDARWIN as a comparison algorithm to support of web-service evolution tasks. Using a set of models to represent the service interfaces (whether this is WSDL or WADL) and to capture their differences, WSDARWIN perform efficient, scalable and accurate comparisons. Furthermore, the results of these comparisons are in a structured format that can potentially be used by other tools such as automatic client adaptation processes. The comparison method is precisely defined by a set of rules based on the representation and delta models. The usage of WSDARWIN was demonstrated on a WSDL and a WADL web service.

Using WSDARWIN we extended our previous empirical study on the evolution of several families of quite widely used commercial web services: Amazon EC2,

FedEx Rate, Bing, PayPal and FedEx Package Movement Information. We examined what types of changes occur in the interfaces of actual, commercial web services and how these changes affect their client applications. Our main observation was that for the most part, as expected, web services were expanded rather than being changed or having their elements removed. This is because the addition of new features does not impact the behavior of clients that already use the service. Furthermore, changes, if made in a conservative manner, do not negatively impact clients much. On the other hand, deletion of elements should be avoided, as it will likely break a client application.

The most important result of the study was to identify a set of frequently applied changes and classify them in three categories according to how they can be handled by the client: *n*o-effect, where changes don't affect the client at all, *non-recoverable*, where changes affect the functionality but cannot be addressed automatically and *adaptable*, where changes affect the interface of the service and the client can be automatically adapted to these changes.

In the future, we plan to extend our comparison method in two directions. The first direction involves identifying more complicated edit operation that consist of the simple ones, change, add, delete and move. This will help us characterize the changes from version to version according to our classification and easily assess their impact on client applications. Second, having defined separate models to represent WSDL and WADL service interfaces, we plan to merge the two into a single web service meta-model to describe service interfaces regardless of their specification. Since the rules and the comparison process are independent of the model, a unified model will allow us to compare any kind of service interface, even heterogeneous once.

Acknowledgments.

The authors would like to acknowledge the generous support of NSERC, iCORE, and IBM.

References

1. Andrikopoulos, V., Benbernou, S., Papazoglou, M.P.: Managing the evolution of service specifications. In: CAiSE '08, pp. 359–374. Springer-Verlag, Berlin, Heidelberg (2008)
2. Aversano, L., Bruno, M., Penta, M.D., Falanga, A., Scognamiglio, R.: Visualizing the Evolution of Web Services using Formal Concept Analysis. 8th International Workshop on Principles of Software Evolution pp. 57–60 (2005)
3. Chawathe, S.S., Rajaraman, A., Garcia-Molina, H., Widom, J.: Change Detection in Hierarchically Structured Information. ACM Sigmod Internation Conference on Management of Data pp. 493–504 (1996)
4. Elio, R., Stroulia, E., Blanchet, W.: Using interaction models to detect and resolve inconsistencies in evolving service compositions. Web Intelli. and Agent Sys. **7**(2), 139–160 (2009)

5. Fluri, B., Würsch, M., Pinzger, M., Gall, H.C.: Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction. IEEE Transactions on Software Engineering **33**(11), 725–743 (2007)
6. Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A.: An empirical study on web service evolution. In: ICWS 2011, pp. 49 –56 (2011)
7. Fokaefs, M., Stroulia, E.: Wsdarwin: Automatic web service client adaptation. In: CASCON '12 (2012)
8. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring Improving the Design of Existing Code. Addison Wesley, Boston, MA (1999)
9. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, 1 edn. Addison-Wesley Professional (1994)
10. Kelter, U., Wehren, J., Niere, J.: A Generic Difference Algorithm for UML Models. Software Engineering 2005, Fachtagung des GI-Fachbereichs Softwaretechnik pp. 105–116 (2005)
11. Pasquale, L., Laredo, J., Ludwig, H., Bhattacharya, K., Wassermann, B.: Distributed cross-domain configuration management. In: Proceedings of the 7th International Joint Conference on Service-Oriented Computing, ICSOC-ServiceWave '09, pp. 622–636 (2009)
12. Ryu, S.H., Casati, F., Skogsrud, H., Benatallah, B., Saint-Paul, R.: Supporting the Dynamic Evolution of Web Service Protocols in Service-Oriented Architectures. ACM Transactions on the Web **2**(2), 1–46 (2008)
13. Wang, S., Capretz, M.A.M.: A Dependency Impact Analysis Model for Web Services Evolution. IEEE International Conference on Web Services pp. 359–365 (2009)
14. Xing, Z.: Model Comparison with GenericDiff. 25th IEEE/ACM International Conference on Automated Software Engineering pp. 135–138 (2010)
15. Xing, Z., Stroulia, E.: Analyzing the Evolutionary History of the Logical Design of Object-Oriented Software. IEEE Transactions on Software Engineering **31**(10), 850–868 (2005)
16. Xing, Z., Stroulia, E.: Refactoring Detection based on UMLDiff Change-Facts Queries. 13th Working Conference on Reverse Engineering pp. 263–274 (2006)
17. Zhang, K., Shasha, D.: Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal on Computing **18**, 1245–1262 (1989)

2.4 WSDarwin: automatic web service client adaptation.

Fokaefs, M., Stroulia, E., 2012. WSDarwin: automatic web service client adaptation. In: Conference of Center for Advanced Studies (CASCON 2012). pp. 176-191.

WSDarwin: Automatic Web Service Client Adaptation

Marios Fokaefs*and Eleni Stroulia†

Department of Computing Science
University of Alberta
Edmonton, AB, Canada
{fokaefs,stroulia}@ualberta.ca

Abstract

The service-oriented architecture paradigm prescribes the development of systems through the composition of services, *i.e.,* network-accessible components, specified by (and invoked through) their WSDL interface descriptions. Systems thus developed need to be aware of changes in, and evolve with, their constituent services. To support this coevolution process, we have developed WSDarwin, a toolkit that facilitates both providers and clients in the evolution of service-oriented systems.

In this work, we focus primarily on the comparison of service-interface versions, in order to precisely recognize their differences, and the adaptation of client applications. We propose a lightweight model to represent service interfaces, an efficient and accurate comparison method whose output can be seamlessly consumed by the adaptation process, a classification of changes in service interfaces based on their impact on client applications and, finally, a generic adaptation algorithm that can be applied for any type of change and on any client regardless of the implementation technology. We demonstrate this part of the WSDarwin toolkit on a client application invoking several versions from the Amazon EC2 web service and we report on the challenges we faced.

1 Introduction

Service-system evolution and maintenance is an interesting variant of the general software-evolution problem. On one hand, the problem is quite complex and challenging due to the fundamentally distributed nature of service-oriented systems, whose constituent parts may reside not only on different servers but also across organizations and beyond the domain of any individual entity's control. On the other hand, since the design of a service-oriented system is expressed in terms of the interface specifications of the underlying services, the overall system needs to be aware of only the changes that impact these interface specifications; any changes to the service implementations that do not impact their interfaces are completely transparent to the overall system. In effect, the WSDL specifications of the system's constituent services serve as a boundary layer, which precludes service-implementation changes from impacting the overall system. Of course, information hiding can still raise certain challenges. For example, even if a change is transparent to the client application with respect to the service interface, its functionality

*The author is holding a CAS Research scholarship.

†The author is a Visiting Scientist with IBM Canada CAS Research, Markham, Ontario, Canada.

can still be affected and cause disruptions to the normal function of the client application, which requires additional effort from the client to adapt the application to the new version of the service. Furthermore, without access to the source code of the service, the client cannot have a full understanding and reason about the changes, which is a crucial aspect of the adaptation process.

Service providers may not precisely know by whom, how often or by how many client applications their services are used. And although changes will always happen (so that providers can improve and extend their offerings), the service provider needs to be somewhat aware of the impact of a specific change on existing clients, in terms of the time and effort necessary to update client software as well as the potential business costs (*e.g.,* when changes cause disruptions to the operations of important partners). Even if the impact is low, the service provider might still need to support some backward compatibility. If the impact is high but the change is still necessary, then it would be desirable to provide appropriate tools to help client developers address the impact of the change. These issues can greatly affect the business aspect of the service provider, since decisions on the evolution of web services are of a very sensitive nature. Indeed, if the impact to an application from a change to the service is unbearable for the client, with respect to cost and effort, this client may opt to switch providers, thus depriving the original provider from the corresponding income.

For these reasons, it becomes evident why recognizing the changes to the specification of a service interface and their impact on client applications is highly desirable and a prerequisite for dealing with the change, whether on the server or on the client side. Assuming that such a precise method for service-specification changes existed, it would be extremely useful if one could (a) characterize the changes in terms of their complexity, and (b) semi-automatically develop adapters for migrating clients from older interface versions to newer ones. Furthermore, by systematizing the process of identifying and addressing the changes to a service, the adaptation cost is reduced and the provider's decision process for the evolution of a service becomes better informed.

In our work, we are building *WSDarwin* a toolkit to support the evolution of web service systems both from the perspective of the client as well as that of the provider. We propose that such a toolkit should provide a series of automatic and semi-automatic techniques to identify faults in client applications due to changes on the consumed service, compare service interfaces, identify and classify changes according to their impact on client applications, automatically adapt the applications to the new version of the service and facilitate the decision process of the service provider when concerning the evolution of the system.

Building on our previous work [6], where we studied the evolution of commercial web services, in this paper we focus on the comparison of service interfaces and the adaptation of client applications. Based on our past experience with developing VTracker [6, 11], a generic tree-differencing algorithm, we propose a service-interface differencing technique based on a lightweight model of the service interface; with this domain-specific method, we are able to improve the efficiency of the process that now produces an output that can be seamlessly integrated with the adaptation process. For the adaptation, we have developed an algorithm to adapt client applications to new versions of web services. This algorithm is general in that it can be applied on any client in spite of its implementation technology (e.g., Java, C++, BPEL etc.) in the presence of any type of change. Moreover, we extend the findings of our empirical study with a classification of changes based on their impact on client applications in order to facilitate the reasoning and understanding from the client's part even in the absence of sufficient information. Finally, we demonstrate the application of the comparison and adaptation techniques on a client application invoking the Amazon EC2 service (as studied in our previous work).

The rest of the paper is organized as follows. In Section 2 we review the related literature focusing mostly on service evolution, client-adaptation techniques and self-adaptive systems. In Section 3 we provide an overview of a service evolution support toolkit as it is being implemented in WSDarwin. In Section 4 we

describe the comparison method used to identify the differences between service interfaces and in Section 5 we discuss how the results of this comparison can be used by a generic adaptation algorithm for client applications. In Section 6 we present the application of the comparison and the adaptation process on a client that invokes the various versions of the Amazon EC2 service and we present details about the challenges we faced in this case study. Finally, Section 7 summarizes this work and concludes with its main contributions.

2 Related Work

Our work in this paper relates to service evolution and client adaptation. In the context of the first topic, which is the general concern of the WSDarwin toolkit, we review research on the evolution of service systems and methods and tools developed to compare service interfaces. In the second topic, we discuss methods for adapting, or facilitating the adaptation of, client applications to evolved services.

2.1 Service-Evolution Analysis

Wang and Capretz [13] proposed an impact-analysis model as a means to analyze the evolution of dependencies among services. By constructing the intra-service relation matrix for each service (capturing the relations among the elements of a single service) and the inter-service relation matrix for each pair of services (capturing the relations among the elements of two different services) it is possible to calculate the impact effect caused by a change in a given service element. These relation matrices can be employed to support service change operations, such as the addition, deletion, modification, merging and splitting of elements.

Aversano et al. [2] proposed an approach, based on formal concept analysis, to understand how relationships between sets of services change across service evolution. To this end, their approach builds a lattice upon a context obtained from service description or operation parameters, which helps to understand similarities between services, inheritance relationships, and to identify common features. As the service evolves (and thus relationships between services change) its position in the lattice changes, thus highlighting which are the new service features, and how the relationships with other services have been changed.

Ryu et al. [10] proposed a methodology for addressing the dynamic protocol-evolution problem, which is related with the migration of ongoing instances (conversations) of a service from an older business protocol to a new one. They developed a method that performs change-impact analysis on ongoing instances, based on protocol models, and classifies the active instances as migrateable or non-migrateable. This automatic classification plays an important role in supporting flexibility in service-oriented architectures, where there are large numbers of interacting services, and it is required to dynamically adapt to the new requirements and opportunities proposed over time.

In a similar vein, the WRABBIT project [5] proposed a middleware for wrapping web services with agents capable of communication and reflective process execution. Through their reflective process execution, these agents recognize run-time "conversation" errors, i.e., errors that occur due to changes in the rules of how the partner process should be composed and resolve such conversation failures.

Pasquale et al. [8] propose a configuration-management method to control dependencies between and changes of service artifacts including web services, application servers, file systems and data repositories across different domains. Along with the service artifacts, Smart Configuration Items (SCIs), which are in XML format, are also published. The SCIs have special properties for each artifact such as host name, id etc. Interested parties (like other application servers) can register to the SCIs and receive notifications for changes to the respective artifact by means of ATOM feeds and REST calls. Using a discovery mechanism the method is able to identify new, removed or modified SCIs. If a SCI is identified as modified, then the discovery mechanism tracks the differences between the two items and adds them as entries in the new SCI.

The aforementioned research works mainly focus on the evolution of inter-dependencies

among services or the evolution of business protocols. On the other hand, our approach focuses on the evolution of the elements within a single service and their intra-dependencies. Furthermore, our approach investigates the effect of service evolution changes on client applications.

Andrikopoulos et al. [1] propose a service evolution management framework. The framework generally aims to support service providers evolve their services. It contains an abstract technology-agnostic model to describe a service system in its entirety, specifying all artifacts such as service interfaces, policies, compositions etc. and divide the artifacts in public and private. This division implies that the management framework has knowledge about the service's back-end functionality, which in turn means that it can be used only by the provider. The authors also propose a classification for the changes based on the basic operations (additions, deletions etc.) and guidelines on how to evolve, validate and conform service specifications to older versions. Although such a management framework may lead to a smooth evolution process, inconsistencies may still occur between services and their clients. Therefore, support to clients is equally important.

2.2 Client adaptation

Benatallah et al. [3] present a methodology for the systematic definition of web-service protocol adapters, in the form of BPEL processes. The proposed techniques aim to facilitate service providers in order to tackle issues such as the evolution of web services, the heterogeneity of interfaces and protocols and the high number of clients with variations in supporting technologies and eventually achieving a certain degree of interoperability. In order to facilitate the creation of adapters, the authors use common mismatch patterns.

In comparison to our work which aims at developing client-side adapters, this one tries to systematically create adapters for incompatible web services but on the server side. There are several benefits on creating the adapter on the provider's side. First, thanks to the availability of source code the provider is in better position to understand the change and create the adapter accordingly. Second, in such a case only one adapter has to be created which can be used by multiple clients. On the other hand, changes may have different effect on client applications depending on their technology. If the change is not completely transparent to the client, more adaptations may be necessary at the client side. In other words, client and provider adapters seem to be complementary solutions rather than mutually exclusive.

Ponnekant and Fox [9] present a set of tools to support service substitutability to allow applications to switch between services from different vendors seamlessly. According to the authors, services can be functionally similar, and thus valid substitutes for each other, under two scenarios: *autonomous services*, where the vendors develop the services independently or *derive services*, where the vendors build on an existing service from a dominant/popular vendor. Obviously, services in the second scenario have more chances of being interchangeable. Incompatibilities between such services can be either structural, semantic or they can concern values or encoding properties. The proposed techniques aim to guarantee the SV-compatibility (structural-value) between two services. Both static and dynamic analyses are employed to resolve the incompatibilities and their criticality is determined manually. Finally, a cross-stub (like an adapter) is semi-automatically generated and the developer is asked to resolve some incompatibilities manually using guidelines provided by the tool.

In this work, the authors employ usage information (i.e., what part of the service is used by the client application) in order to deal only with the relevant incompatibilities. Although filtering out irrelevant incompatibilities for the *current* version of the application might improve the performance of the tool, it will eventually hinder the extensibility of the application since, if the application was to expand by also using other parts of the service, the client-development team will have to address the incompatibilities again. WSDarwin addresses all incompatibilities regardless of whether they are used or not. After all, Axis2 would generate a client proxy for the whole service interface, thus, potentially the entirety of the service can

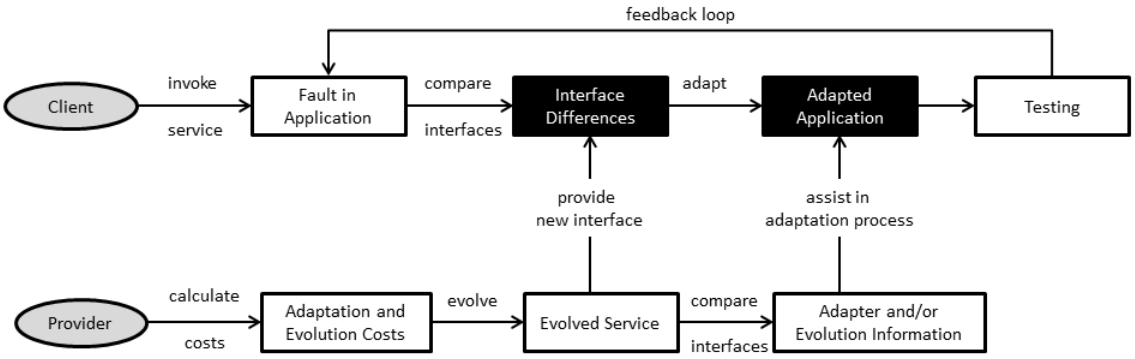


Figure 1: The WSDarwin service evolution support toolkit. This paper focuses on the comparison and adaptation parts (black boxes).

be used.

Both of these works deal mostly with the problem of service interchangeability across different vendors. This is related to the problem of client adaptation to new versions of a single service, but it also presents some different challenges. First, when we have different vendors, it is expected to have differences in the semantics of the services. For example, two services might use the terms "search" and "find" for the same task. This might make the mapping process especially challenging. Second, in case the provider opts to offer an adapter from the server side, it is necessary to know what the alternative services from the other vendors are. Also, if a new alternative service enters the market, the adapter has to be updated. Another difference between these works and ours is that incompatibilities have to be identified manually or at least with manual intervention. On the other hand, WSDarwin has a completely automatic comparison component.

Villegas et al. [12] propose a framework to support the (self-)adaptation of systems using quality and adaptation properties as adaptation goals and adaptation metrics to evaluate these properties. The framework evaluate properties concerning the structural, behavioral and functional aspect of the system, such as stability, robustness, performance etc. In our work, we focus more on the structural adaptation of the client applications, however, we plan to expand our work to cover more of the behavioral properties described in this work.

3 Overview of WSDarwin

The WSDarwin toolkit supports the evolution of service systems in a provider-client ecosystem. Such an ecosystem has two central parties: a provider, who controls the web service and is responsible for evolving it, and the client, who controls the client application and has to catch up with the evolving web service. The key artifact exchanged between the two parties is the service interface which is the only communication between the provider and the client and they will have to perform all evolution activities solely using the information on the interface.

Figure 1 depicts the evolution process of a service system in the described ecosystem. The role of the provider starts at the early stages of the evolution. First and foremost the provider has to study the environment to confirm that there is a need for a change and that the time is ripe. In general, the decision for a change involves not only technological properties and constraints, but it also includes business and economic concerns. After the evolution of the service, the provider has to publish the new interface so that it becomes accessible to the clients. Depending on the environment, the provider may opt to facilitate the adaptation process of the client by providing adapters [3, 9] and evolution information [4]. However, these adapters or the information have to be provided in a format that can be understood and consumed by tools.

In the event of an evolved service, the client

has first to recognize the change. This can be practically assessed by invoking the service anew and establishing that the normal function of the client application is disrupted (e.g. an exception was thrown or the results were not the expected ones). Then, the client has to retrieve the interfaces that correspond to both the old and the new version of the service and compare them in order to identify the changes. The result of this comparison can be further analyzed in an attempt to identify the purpose behind the changes. This can be helpful and save time and effort for the developers because the nature and the purpose of a change is directly related with invoking applications. For example, if a change is characterized as a refactoring, it may affect the interface of the service but there is high confidence that the functionality of the service may remain unaffected. Furthermore, the result of the comparison should be in a format that it is appropriate to be effectively utilized by the adaptation process. The adaptation of client application is usually tied to the specific implementation technology, but there are still some quality and syntactic rules that should hold. Moreover, the provider can facilitate the adaptation process as described previously. The final step is for the client to test the application and confirm that everything works as expected and if there is a need for further modifications.

4 Comparison of Web-Service Interfaces

As we have mentioned a comparison method should adhere to three quality properties:

- Accuracy
- Efficiency/Scalability
- Systematized output (which can be used by other tools)

Our previous comparison method, VTracker [6], is a generic differencing algorithm that can be used to compare heterogeneous interfaces, i.e., interfaces described in different schemas. For this reason, this method uses fuzzy mapping and partial matching. In the first case, since we don't always know a mapping between the elements of the two interfaces, the algorithm compares all elements with each other (regardless of their type) and establishes a mapping based on their structural similarity. In the second case, the algorithm uses the notion of distance to compare elements with each other. This can cause problems in the case of elements of different types named in a similar manner if they correspond to the same concept. In the case of web services, the convention is to name operations and their input and output types similarly to denote their relationship. Fuzzy mapping and partial matching maybe the reasons for efficiency and accuracy issues. However, if we imbue the comparison process with knowledge about the structure of the interfaces we can significantly improve these quality properties.

In the WSDarwin comparison method, we ensure efficiency by using a reduced, domain-specific model to represent the syntactic information of a service interface. The interfaces are parsed and a model representation is created for each one of them on which the comparison method is applied. The model captures the most important information of a service's elements such as names, types, their structure and the relationships with each other, thus, providing a simpler syntactical representation of the service representation than WSDL and making it more lightweight. The simplicity of the model allows for improved efficiency. Moreover, we employ certain heuristics on name comparisons to further improve the efficiency. The reason for that is that within the same service (even between versions) names can be treated as unique and therefore as IDs. The use of the same name for different elements is not likely (and in many cases it is not allowed). For this reason, it only makes sense to compare strings using exact matching and not partial matching techniques such string edit distance.

This model also ensures accuracy. Instead of comparing named XML nodes, we compare model entities based on their specific type (e.g. operations with operations, complexTypes with complexTypes etc.). This way it is not necessary to compare all elements against each other, thus avoiding false results due to

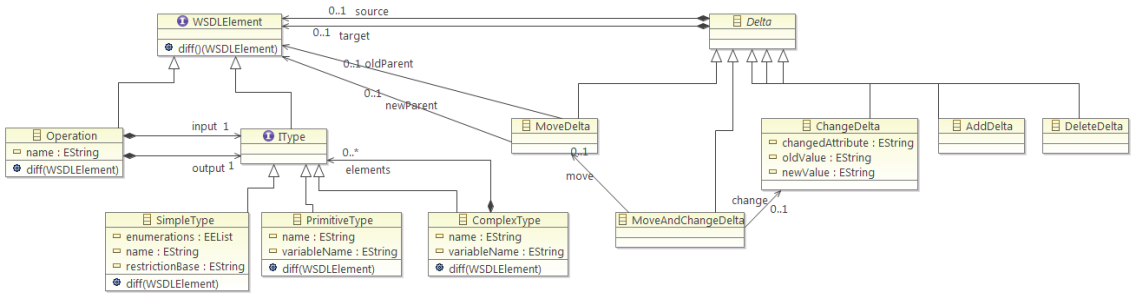


Figure 2: The Delta model used for the comparison output. The leftmost hierarchy represents the service interface.

fuzzy mapping and improving the efficiency.

As far as the output is concerned, it is produced using the model shown in Figure 2. The rightmost hierarchy in the figure corresponds to the model used in the comparison. Essentially, the operations, which are the invocation points between the provided service and the client application, have input and output types. The type hierarchy is in accordance with the XML Schema specification[1]: `PrimitiveTypes`, such as strings, integers, boolean etc., `SimpleTypes` are types that pose certain restrictions on their values (e.g. enumerations) and `ComplexTypes` which contain other types. To make the model simpler than the input interface we omitted some elements that added no additional structural information as far as clients are concerned. We omitted `messages` and high level `elements` from the schema, which only serve as references. Therefore, only the elements to which these references point were eventually included in the model. The rightmost hierarchy models the changes. We can have different types of deltas including changes, additions, deletions, moves and moves and changes. The two hierarchies are connected through the Bridge design pattern [7] and their relationship is that each delta has a source WSDL element and a target WSDL element.

In our comparison, we used some rules to map and differentiate the elements between different versions of the service interfaces. For the definition of the rules we use the following notation based on the model we described above. For each version $v \in V$ we extract the following sets:

- E_v: The set of WSDL elements of the service. This set contains tuples (id, t, a, s) where id is the identifying attribute of the element (usually the name), t is the type of the element (complex type, primitive type, simple type or operation), a is the set of other attributes and s is the structure of the element. This set is the superset of the WSDL elements, i.e., $CT_v \cup PT_v \cup ST_v \cup O_v = E_v$, where
  - CT_v: The set of complex types of the service. This set contains tuples (n, v, t), where n is the name of the complex type, v the name of the corresponding variable (XSD element) in the WSDL file and t is the set of types it contains.
  - PT_v: The set of primitive types of the service. This set contains tuples (n, v), where n is the name of the primitive type (string, integer, etc.) and v the name of the corresponding variable (XSD element) in the WSDL file.
  - ST_v: The set of simple types of the service. This set contains tuples $(n, r, enum)$, where n is the name of the simple type, r is the type of the restriction base and $enum$ is the set of values for the enumeration.
  - O_v: The set of operations of the service. This set contains tuples (n, it, ot), where n is the name of the

[1] http://www.w3.org/XML/Schema

operation, *it* is the input type and *ot* is the output type.

- A_e: The set of attributes, other than the identifying one, for an element $e \in E$.
- S_e: The structure of a complex element $e \in E$. The structure refers to the children of complex elements such as input and output types for operations and elements for complex types.

Finally, for each comparison Δ, between two versions $v_1, v_2 \in V$, we determine the added, deleted, changed and matched elements by using the symbols "+", "-", "*" and "=" respectively. Therefore, E^+_Δ is the set of elements that were added. We also use the symbol "#" to denote mapped elements, e.g. $E^\#_\Delta$.

Table 1 summarizes the rules we use to compare service interfaces.

1. The first rule is the exact matching rule. In case of simple attributes (such as the element's ID and attributes belonging in the A_e set of the element), two attribute values are the same if and only if they have the same literal. In case of structure (i.e. the set of children of an element), two elements are considered structurally equal if and only if their children are equal. Children equality is determined in an iterative manner.
2. The second rule states that two elements are mapped together if their type and at least one of their identifying attributes, name and structure, match.
3. An element is considered changed if its name (its ID) was found in both versions (i.e., it is mapped between the two versions) but there were some changes in the values of other attributes.
4. If there is a change in the structure of the element (i.e., its children), the change is propagated from the nested element to the parent, even if the parent is not directly changed. This is because the adaptation process starts from the root element of a service request which is considered to be the operation. Therefore, if some part of its input or its output is affected the operation is still considered affected.
5. If two elements are mapped and no differences are identified, they are label as matched. The reason to retain matched elements is that the adaptation process needs a full mapping between the two versions as we will see next.
6. An addition is identified if an element's name (its ID) that existed in the old version was not found in the new version.
7. Correspondingly, a deletion is identified if an element's name did not exist in the new version but was found in the old version.
8. In a second phase, the additions and deletions are reanalyzed to identify changes in the IDs or moves. If an element is identified as deleted from the old version and another element as added in the new version and the two elements have identical structure but differ with respect to their IDs then these elements are labeled as changed (renamed).
9. In a similar scenario, where elements are mapped between the deleted and added sets, these elements are marked as moved. The reason they couldn't be identified in the first run of the comparison is because the process follows the structure of the service interface and elements are compared only with respect to their parents.
10. If the moved elements also differ in their structures or their IDs, they are labeled as moved and changed. If they differ with respect to both structure and ID, then they are considered different elements and are report as an addition and a deletion.

4.1 Service change classification

In our previous empirical study [6] we have reported several change scenarios in service interfaces some of them we actually observed in the studied services, some others were based on our experience on software-maintenance activities. In this work, we provide a classification of such changes based on their potential

Table 1: The definition of rules used by WSDarwin for the comparison of web service interfaces.

	Name of comparison rule	**Rule**
1	Exact matching	$\forall a_{e_1} \in A_{e_1}, \forall a_{e_2} \in A_{e_2} : a_{e_1}.literal = a_{e_2}.literal$
2	Mapping	$\exists e_1, e_2 \in E^{\#}_{\Delta} : e_1.t = e_2.t$ and ($e_1.id = e_2.id$ or $e_1.s = e_2.s$)
3	Changed	$\exists (id_i, t_i, a_i, s_i) \in E^{\#}_{\Delta}$ and $\exists (id_j, t_j, a'_j, s_j) \in E^{\#}_{\Delta}$
4	Propagated change	$\exists (id_i, t_i, a_i, s_i) \in E^{\#}_{\Delta}$ and $\exists (id_j, t_j, a_j, s'_j) \in E^{\#}_{\Delta}$
5	Matched	$\exists (id_i, t_i, a_i, s_i) \in E^{\#}_{\Delta}$ and $\exists (id_j, t_j, a_j, s_j) \in E^{\#}_{\Delta}$
6	Added	$\exists e_{v_2} \notin E^{\#}_{\Delta}$
7	Deleted	$\exists e_{v_1} \notin E^{\#}_{\Delta}$
8	Changed (Renamed)	$\exists (id_i, t_i, a_i, s_i) \in E^{-}_{\Delta}$ and $\exists (id'_j, t_j, a_j, s_j) \in E^{+}_{\Delta}$
9	Moved	$\exists (id_i, t_i, a_i, s_i) \in E^{-}_{\Delta}$ and $\exists (id_j, t_j, a_j, s_j) \in E^{+}_{\Delta}$
10	Moved and Changed	$\exists (id_i, t_i, a_i, s_i) \in E^{-}_{\Delta}$ and $\exists (id'_j, t_j, a'_j, s'_j) \in E^{+}_{\Delta}$

impact on client applications. According to this classification, we distinguish three types of changes: (a) no-effect, (b) adaptable and (c) non-recoverable.

No-effect changes do not impact the client at all. The client functionality is not disrupted and neither is the interface, which practically means that the client can still operate using the old stub (i.e., the service proxy auto-generated locally in the client). An example of such a change is the addition of new functionality (provided that there is no dependency with the old functionality).

Adaptable changes affect the interface of the client, but the functionality of the service remains the same. Viewed from the point of view of the service provider, these changes usually correspond to refactorings on the source code of the service. In other words, they are changes meant to improve the design of the service and leave the functionality unaffected. They can be easily addressed by generating a new stub and changing the old stub, still used by the client application, to invoke the new one and thus the evolved service. This way we avoid changing the client code by modifying only autogenerated code. Refactorings are representative instances of adaptable changes. For example, in Amazon EC2 we had the application of an "Inline Type" refactoring (i.e., the elements of a type were redistributed to the type's parent and the said type was deleted). In this case, the change is adaptable since in the new version we have exactly the same data but only packaged differently.

Non-recoverable changes imply that the functionality of the service is affected, in a way that the client breaks and we cannot address the issue without changing and recompiling the client code. In some cases, the change is so subtle as not to affect the interface of the client. In other words, the client still works but the results produced are not the desired ones. The problem in this case can be identified by means of unit and regression testing.

Being able to recognize the nature of the change in these terms can save a lot of effort on the developer's part. For example, in case of no-effect changes there might be no real need to proceed with the adaptation of the client application. On the other hand, in non-recoverable changes, the adaptation of the client to the changed interface may actually accomplish nothing, so it can be skipped and proceed directly to the complete reevaluation and reengineering of the client.

5 Adapting clients to changes

Using the results of our previous empirical study with respect to frequent change scenarios we propose an algorithm to automatically adapt clients to changed service interfaces in case of adaptable changes. Algorithm 1 demonstrates the necessary steps.

Once it has been confirmed that the invoked service has changed, the task becomes to generate the new stub (step 1). This will be

Algorithm 1 Client Adaptation

```
1: Generate the stub for the new version of the service.
2: Add a reference of the new stub in the old stub.
3: Instantiate the new stub in the constructors of the old stub, by invoking the corresponding constructor
   and using the appropriate parameters.
4: Find a complete mapping for the types and operations between the two versions and identify what
   changed and how.
5: for all the changed operations do
6:    Delete the body of the corresponding method of the old stub.
7:    //In the body of the method of the old stub, prepare the input and the output for the new method.
8:    //For the input:
9:    Create an instance for the input of the new version of the operation.
10:   for all the elements that match between the old and the new input do
11:      if an element is an object then
12:         Create an instance of the object using types from the new stub.
13:      end if
14:      Copy the values of all primitive elements (integers, strings etc.) from the old input and its objects
         to the new input and its objects.
15:   end for
16:   if the new input has new elements then
17:      Assign default values to the new elements ('0' if numeric, empty string if text and null if object).
18:   end if
19:   //For the output:
20:   Create an instance of the old output.
21:   Using the new input invoke the new version of the operation from the new stub.
22:   for all the elements that match between the old and the new output do
23:      if an element is an object then
24:         Create an instance of the object using types from the old stub.
25:      end if
26:      Copy the values of all primitive elements (integers, strings etc.) from the new input and its objects
         to the old input and its objects.
27:   end for
28:   if the old input has deleted elements then
29:      Assign default values to the deleted elements ('0' if numeric, empty string if text and null if object).
30:   end if
31:   Return the old output.
32: end for
```

used, from now on, to support the communication between the old client and the new service. However, the new stub cannot be used directly (as this would imply changing the client code); instead, the old stub will be automatically changed to invoke the new stub. Next, we use the diff script produced by our comparison method that contains the full mapping between the types and the operations of the old version of the service interface and the new version. This will ensure that the changed elements are fully reconstructed and accurately replaced in the source code. Finally, it is just a matter of changing the old stub's methods to appropriately call the methods of the new stub. It's important to note that the client provides input in the old format and also expects the result to be in the old format. Therefore, the new input has to be constructed using values from the old input and the new method may be invoked to obtain its output. Then using values from this output, an output of the old format is constructed and returned to the client.

In general we can make two observations for this algorithm. The first is about its generic nature. Firstly, it does not depend on the nature of the change. If we consider an operation in its fundamental form, as a mathematical function, it's a mechanism that takes an input and it produces an output. If the data in the input and the output is not affected by the change (adaptable change) then we can reengineer them so that the change is transparent to the client. As a consequence all adaptable changes can be viewed by protocol changes (that primarily affect the communication between two entities). Furthermore, the general concept of data transformation can be applied on all kinds of clients regardless of their specific implementation technology. How these data transformations are performed by different technologies and what kind of challenges this entails indeed depend on the specific technology and will be discussed in the next section in the case of a Java client.

The second observation concerns the degree of intrusion of the adaptation in the general development process of the client applications. The proposed algorithm only changes the client stub. Since this part of the code is automatically generated the developers have very little knowledge about it and it contributes nothing to the developers' general awareness about the application. Therefore, by changing only the stub and making the change look transparent from the client's perspective, we maintain the developers' awareness. Figure 3 shows how the adaptation is applied in this manner. Naturally, in this scenario the adaptation process takes place only on the client side.

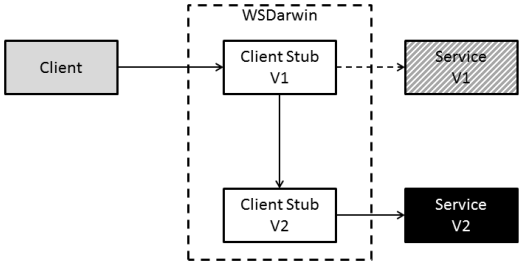


Figure 3: Adaptation process.

6 Amazon EC2: a case study for WSDarwin

In this section, we present the application of the comparison and adaptation processes of WSDarwin on a Java client that invokes several versions of the Amazon EC2 web services, as this was described in our previous work. The case study is presented under certain assumptions.

First, we only focus on these two parts of the evolution process and not on the testing part. This practically mean that whatever change we have to deal with and whatever adaptations we apply, we do not afterward confirm that the functionality of the client is not disrupted. In other words, we view all changes between the versions of the service as adaptable changes; we fix the interface inconsistencies but we assume that the functionality is checked later (manually or automatically). To this end, the only artifacts we need in order to study our tools are the web service interfaces (in the form of WSDL files) and the client stubs corresponding to the various versions of the service.

Second, we assume Apache Axis2 as the web-

service middleware[2]. One important difference between Axis2 and its predecessor Axis is that the former has as a configurable option to control the "wrapping" of types, while the latter "unwraps" the types by default. "Wrapping" a primitive (e.g., an integer or a string) type means that the middleware creates a corresponding complex type to include the primitive as an element and the client generation engine may create a class corresponding to each complex type. For our experiments, we followed the default configuration of Axis2, which wraps the types and does not produce a class for each complex type but rather adds them all as inner classes in the stub. This way the adaptation method has to transform only one compilation unit (i.e., java file).

Finally, for the analysis and transformation of the Java client proxy we used the Java Development Toolkit (JDT) of Eclipse[3] that employs the Abstract Syntax Tree (AST) representation to manipulate Java source code.

We believe that in spite of these assumptions our general conclusions can be transferred and still hold in other configurations and environments as well. However, we have observed that as we specialize the aspects of the adaptation process (technologies, configurations etc.), certain challenges may arise and we will study them as they have appeared in this particular case study.

Figure 4 demonstrates the output of the comparison process between two versions of the Amazon EC2 service. The two versions of the output we present in this figure are to show the difference when the option to detect moves is enabled. As it can be observed, the moved elements are no longer reported as deleted. However, the respective additions of the moved elements to the new parent remain in order to retain the full mapping between the two versions. This is because additions are only visible in the new version and deletions only in the old version. If we removed the additions as well the moved elements would only be visible through the old parent and this could cause problems in the adaptation process.

The script is organized according to the structure of the service interface. The first level concerns the operations. First, the type of the difference is reported (Change, Add, Delete, Move etc.), then the name of the old operation and the name of the new operation. On the second level, we have the input and the output of the operation presented in a similar way. On the third level, we report the elements of the complex types. In case the elements are complex types as well, these are reported in a new level. In the case of changes in attributes of a WSDL element, after we report the names of the old and the new version of the element, the name of the changed attribute is reported along with the old and the new value. In the case of additions, we report only the new version of the element, since the old does not exist. The opposite happens in the case of deletions. In moves, we also report the names of the old and the new parent. In case of moves and changes, we combined the information reported by the individual moves and changes. Another relevant observation related to the figure is that the operation and its input are reported as changed without any more information. This is because this is not an attribute change but rather that this change was propagated from the children of these elements. Therefore, this is to denote that the input and, in turn, the operation itself were affected by these changes.

As far as the adaptation of the client is concerned, we observed certain inconsistencies between the service interface file and the client proxy, mainly because of the configuration and the tool we used to generate the proxy. The first of these inconsistencies concerns the names (IDs) of types and operations. For example, the WSDL for the Amazon EC2 specifies the names of operations starting with an upper case character. However, when this is translated into Java methods, the Java naming conventions have to be followed and the method names start with a lower case letter. This issue can be easily overcome by ignoring the case of the names when comparing them with each other. Another naming issue concerns the elements of `ComplexTypes`, which, in the client stub, would correspond to attributes of classes. Axis2 names these attributes using the `name` attribute of the element and prefixing it with the term "local". Obviously, there is no longer a direct correspondence between

[2]http://axis.apache.org/axis2/java/core/
[3]http://www.eclipse.org/jdt/

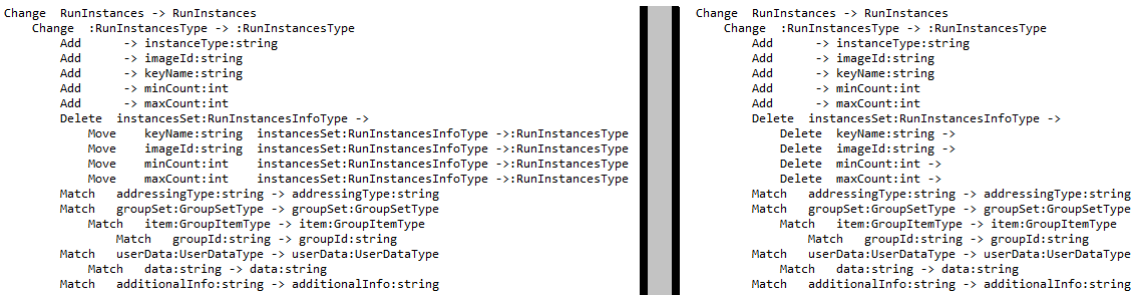


Figure 4: Snippet of the diff script between two versions of the Amazon EC2 service. Left-hand side is with the detection of Move operations, right-hand side is without.

the elements and the attributes. This issue can be mitigated by comparing the terms of the names. Therefore, we tokenize the names into their constituent parts, we compare them using the exact-matching principle and we report the names as equal if the one with the least number of terms is fully included in the other (usually the only term that remains is "local"). In order to increase the confidence that two names are equal in this sense, we also compare their type. Finally, a similar approach is employed to find and use the appropriate public accessors for the local attributes; the names of the elements are partially matched with the name of the accessor methods (with the exception of the "get" and "set" keywords) and a method invocation is created to invoke them where necessary.

The second inconsistency concerns the types of elements and more specifically arrays and collections. In the WSDL specification, arrays of elements are specified by means of additional attributes on the minimum or maximum occurrences of this element in the parent complex type. In this case, the client generator translates the multiplicity in an attribute whose type is an array of the element's type. Once again, the direct mapping is lost (because `Type` is not the same as `Type[]`). In order to compare such cases, we use JDT's type binding to first resolve whether we have to deal with an array and then we get the type of the elements in the array. After the mapping, we also have to address the construction of the new array with the elements from the old array (and all the nested changes). Axis2, in case of arrays, usually generates an adding method as well. Therefore, we can invoke this method iteratively for all elements in the old array. Alternatively, if the adding method is absent, we can construct the new array manually, by setting each element of the old array in the corresponding cell of the new array (i.e., `newArray[i] = oldArray[i]`).

Although the proposed adaptation algorithm is generic and can generally be applied to any client against any change, these inconsistencies show that additional effort and attention is necessary to implement this algorithm in an adaptation tool. However, the algorithm can provide guidelines and a general skeleton for all adaptation tools.

6.1 Implementation status of the WSDarwin adaptation process.

At the time of writing this paper, we are in the process of implementing and perfecting the WSDarwin tools with a particular focus on the automatic adaptation of client application. The toolset is implemented as an Eclipse plug-in and its implementation status is as follows:

1. We have manually applied the adaptation process on 18 versions of the Amazon EC2 web service (as studied in [6]). That is we have identified and studied the changes that occurred between these versions and identified all the challenges in the adaptation process as they were presented in this section.

2. We have applied and tested the tool on educational and simple service systems. It addresses most of the edit operations in

service interfaces and produces adapted clients with no compilation errors.

3. We are currently working in addressing the specific challenges that came up in the Amazon EC2 web service. The goal here will be to produce adapted client proxies with no syntactic error (the functional sanity of the client is out of scope for this work).

7 Conclusion and Future Work

In this work, we introduced *WSDarwin*, a toolkit to support providers and clients alike in the evolution of web-service systems. The set of tools in WSDarwin support

(a) the analysis of the evolution of web services (with a lightweight model for representing WSDL specifications and a corresponding differencing algorithm),

(b) the decision-making process of providers, when considering the evolution of their services (, and

(c) the adaptation of client applications to evolving web services (with an algorithm that modifies the client stub to use the new service interface).

The contribution of this paper is the description of the first and third aspects of the WSDarwin toolkit, as we focus specifically on comparing service interfaces and adapting client applications.

1. We presented a lightweight model to represent the service interface, which is then combined with delta model to represent the differences between subsequent versions of the service interface and systematically produce an output that can be seamlessly consumed by other tools in the WSDarwin toolkit. The comparison methodology is based on a set of well-defined which in combination with the lightweight model allow for greater efficiency, scalability and accuracy. Finally, the format of the comparison output is in direct accordance to the client proxy with enough detail to be directly consumed by the adaptation process.

2. Next, we presented a client adaptation algorithm. This algorithm is generic enough to be applicable on any kind of client application, in spite of its technology, to tackle any kind of change. It contains a complete set of guidelines which can be used to build tools to automatically support the adaptation of client applications for specific environments both from the side of the provider or that of the client.

We have built a prototype tool for the comparison methodology as described in this paper and applied in the Amazon EC2 case study. We plan to expand the comparison tool support to be able to identify more complicated changes than the ones we report in this paper. The ability to identify complex changes would also give the tool the ability to reason about the purpose and the nature of the change and by extension its impact on client application. For example, refactorings, which are a form of complex change, usually do not affect the functional behavior of the system and as such can be easily classified as adaptable changes. Apart from the type of changes, we plan to extend the comparison methodology with respect to the data it examines. A client application can be affected by a change not only on the interface of a service but also on its quality properties. Such information can be found in Service Level Agreements (SLAs). By combining the results from comparing both the interface and the SLAs of a service, we can give a more complete picture of the evolution of a service to its clients.

As for the adaptation process, we have created a prototype tool in the form of an Eclipse plug-in which uses the proposed adaptation algorithm to adapt Java-based clients to changed services based on the configuration described in our case study. We plan to expand our tool to different configurations and possibly create guidelines on how to build technology specific implementations of the adaptation algorithm.

Acknowledgments

The authors would like to acknowledge the generous support of NSERC, iCORE, and IBM.

About the Authors

Marios Fokaefs is a PhD student with the Service Systems Research Group in the Department of Computing Science in the University of Alberta, Canada. He received his BSc from the Department of Applied Informatics in the University of Macedonia, Greece and his MSc from the Department of Computing Science in the University of Alberta, Canada. His research interests include object-oriented and service-oriented design and reengineering.

Eleni Stroulia is a Professor and NSERC/AITF Industrial Research Chair on "Service Systems Management" (w. support from IBM) with the Department of Computing Science at the University of Alberta. Her research addresses industrially relevant software-engineering problems and has produced automated methods for migrating legacy interfaces to web-based front ends, and for analyzing and supporting the design evolution of object-oriented software. More recently, she has been working on the development, composition, run-time monitoring and adaptation of service-oriented applications, and on examining the role of web 2.0 tools and virtual worlds in offering innovative health-care services.

References

[1] Vasilios Andrikopoulos, Salima Benbernou, and Mike P. Papazoglou. Managing the evolution of service specifications. In *CAiSE '08*, pages 359–374, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] L. Aversano, M. Bruno, M. Di Penta, A. Falanga, and R. Scognamiglio. Visualizing the Evolution of Web Services using Formal Concept Analysis. *8th International Workshop on Principles of Software Evolution*, pages 57–60, 2005.

[3] Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R. Motahari Nezhad, and Farouk Toumani. Developing adapters for web services integration. In *CAiSE*, pages 415–429, 2005.

[4] Kingsum Chow and David Notkin. Semi-automatic update of applications in response to library changes. In *ICSM '96*, pages 359–, Washington, DC, USA, 1996. IEEE Computer Society.

[5] Renée Elio, Eleni Stroulia, and Warren Blanchet. Using interaction models to detect and resolve inconsistencies in evolving service compositions. *Web Intelli. and Agent Sys.*, 7(2):139–160, April 2009.

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

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, November 1994.

[8] Liliana Pasquale, Jim Laredo, Heiko Ludwig, Kamal Bhattacharya, and Bruno Wassermann. Distributed cross-domain configuration management. In *ICSOC-ServiceWave '09*, pages 622–636, 2009.

[9] Shankar R. Ponnekanti and Armando Fox. Interoperability among independently evolving web services. In *Middleware '04*, pages 331–351, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[10] S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul. Supporting the Dynamic Evolution of Web Service Protocols in Service-Oriented Architectures. *ACM Transactions on the Web*, 2(2):1–46, 2008.

[11] N. Tsantalis, N. Negara, and E. Stroulia. In *ICSM 2011*, pages 586 –589, sept. 2011.

[12] Norha M. Villegas, Hausi A. Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. A framework for evaluating quality-driven self-adaptive software systems. In *SEAMS '11*, pages 80–89, New York, NY, USA, 2011. ACM.

[13] S. Wang and M. A. M. Capretz. A Dependency Impact Analysis Model for Web Services Evolution. *ICWS 2009*, pages 359–365, 2009.

2.5 The WSDarwin Toolkit for Service-Client Evolution

Fokaefs, M., Stroulia, E., June 2014a. The WSDarwin Toolkit for Service-Client Evolution. In: IEEE Internation Conference on Web Services, Work In Progress (ICWS 2014 WIP). IEEE, Anchorage, Alaska, USA, pp. 716-719.

The WSDarwin Toolkit for Service-Client Evolution

Marios Fokaefs and Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
Email: {fokaefs,stroulia}@ualberta.ca

***Abstract*—As primarily modular and distributed architectures, service-oriented architectures may impose new challenges in software evolution. Since web services evolve independently, this may cause disruptions to the proper function of consuming software. In this paper, we present *WSDarwin* , an Eclipse plug-in to support the evolution of service clients, including (a) identifying the differences between two service versions; (b) automatically adapting the client application to the new version; and (c) testing the client to confirm it functions properly.**

I. Introduction

Web services are independently developed and functionally autonomous pieces of software; therefore systems comprised of web services typically include components outside the domain of control of any single organization. As a result, when web services evolve independently, the functionality of the software systems that rely on them may be disrupted. In these cases, the service clients have to react fast and adapt to the new service versions, typically with limited amount of information.

In this paper, we describe the *WSDarwin* Eclipse plug-in for supporting the adaptation of clients to changed web services. In the event of service evolution, WSDarwin facilitates service-client adaptation by providing three major functionalities: (a) *comparison* of the two versions of the service interface (before and after the evolution) to identify their differences; (b) client *adaptation*, using the service-interface differences as input to automatically adapt Java-based client applications by transforming the client proxy; and (c) *regression testing* of the client's JUnit test cases in order to confirm that the proper function of the client application was not disrupted after the change of the service and the adaptation process.

The rest of this paper is organized as follows. In Section II, we give the overview of the WSDarwin plug-in. In Section III, we demonstrate how the tools can be used on an example from the Amazon EC2 web service [1] and how they perform in terms of execution time. Finally, Section IV concludes this paper.

II. WSDarwin Overview

Several methods have been proposed for adapter construction in the event of service evolution. Benetallah et al. [1] propose BPEL-based adapters to handle the changes on the message exchange protocol between two versions. Ponnekanti and Fox [2] propose a method to create adapters between services from different providers to support interoperability. Finally, Kaminski et al. [3] propose a tool to iteratively create interface adapters to simultaneously support clients that correspond to multiple versions of the service. All these efforts focus on creating adapters on the server side. While this approach is *proactive* in that backwards compatibility is guaranteed by the service provider, WSDarwin follows a *reactive* approach. Since we cannot fully assume that the provider will always make backwards compatible changes, the clients will have to be given tools to effortlessly adapt to the new version. WSDarwin offers three tools for comprehensively supporting the client-adaptation process described as follows.

The **comparison module** can compare both WSDL[2] interfaces for operation-centric services, and WADL[3] interfaces for RESTful services. The first step in the comparison process is to transform the input service interfaces into the WSMeta [4] representation. WSMeta is a lightweight abstraction of both service interface specifications, WSDL and WADL. According to WSMeta, a *Service* is a collection of *Interfaces*, which contain *Operations*. An Operation has an input and an output *ComplexType*. The ComplexTypes contain other ComplexTypes or *SimpleTypes* or *PrimitiveTypes*[4]. The purpose of the WSDL-to-WSMeta (WADL-to-WSMeta) transformation is to strip the interface of purely syntactic elements that do not convey information about the evolution of service, such as XML Schema elements, messages, service bindings and documentation. Eventually, the input service interface is represented as a collection of operations with input and output types. WSMeta also resolves any references in the service interface and replaces them with composition. This reference resolution substantially improves the performance of the comparison process. In fact, we have shown the improvement of this method over XML tree-differencing techniques in our previous work [5].

Once the input interfaces have been translated into WSMeta, the comparison module maps the service elements between the two versions through a parallel depth-first traversal. If two elements have the same identifier, then they are considered the same element in the two versions (i.e., a complete "match"). If two elements have different names, their structure (i.e., their children elements) is checked and, if it is identical, the elements are "mapped", i.e., they are considered to be the same element that has suffered a change in the evolution process, in this case "renaming". If an element of the older (newer) version cannot be mapped in the newer

[1] http://aws.amazon.com/ec2

[2] http://www.w3.org/TR/wsdl

[3] http://www.w3.org/Submission/wadl/

[4] A PrimitiveType can be an integer, a float, a double, a character, a boolean or a string. A SimpleType is a PrimitiveType with special restrictions, for instance, an enumeration. Definitions are according to the W3C XML Schema specification http://www.w3.org/XML/Schema.html

978-1-4799-5054-6/14 $31.00 © 2014 IEEE
DOI 10.1109/ICWS.2014.113

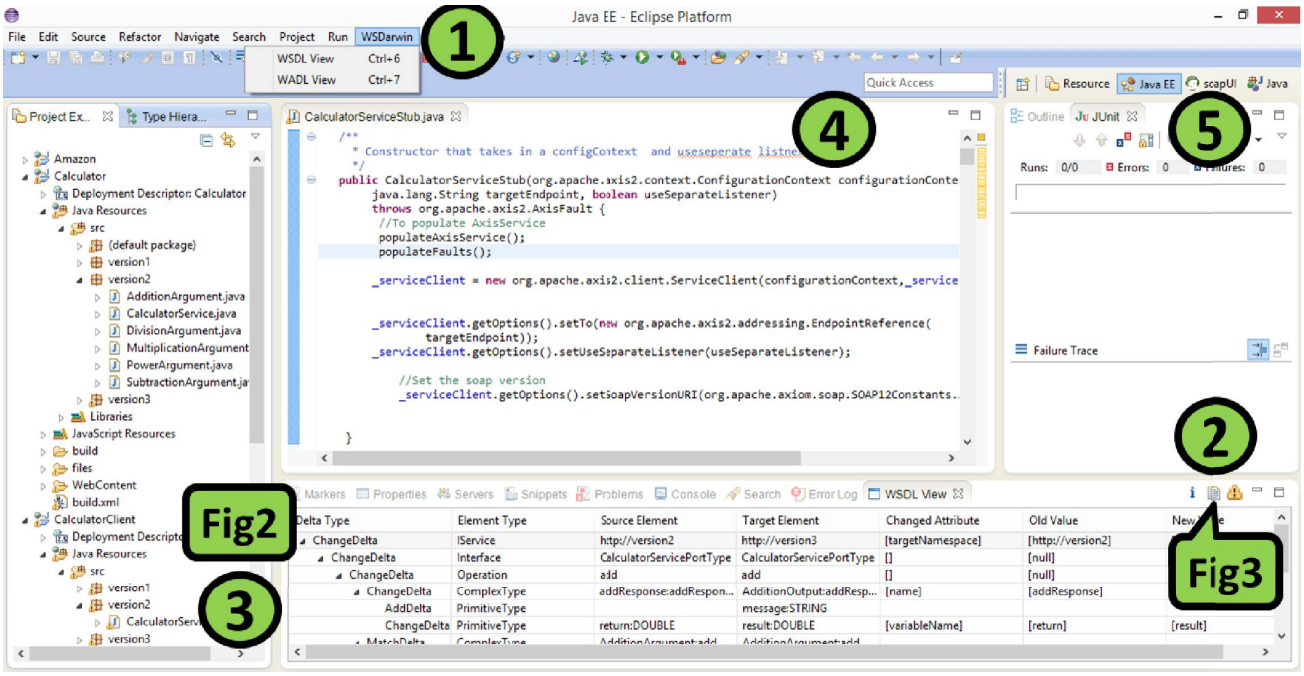


Fig. 1. The WSDarwin interface in the Eclipse workspace.

(older) one, it is considered "removed" (newly "added"). After the interface-traversal process has reached the leaf interface elements, the traversal process is reversed, and the precise *deltas* (i.e., units of difference) are constructed between the mapped elements (taking into account the deltas of their children elements). In this process, the tool distinguishes between complex elements that have children like Service, Interface, Operation and ComplexType and simple elements that do not have an internal structure like SimpleType or PrimitiveType. WSDarwin recognizes six types of deltas: (1) addition, (2) deletion, (3) change, (4) move, (5) move and change and (6) match (no change), as follows.

- If a simple element appears with the same name and the same type in the new version, it is marked with a *match* delta.
- If a complex element appears with the same name and all its children elements have been marked as *matched*, it is also marked with a *match* delta.
- If a complex element has a different name but matched children, it is marked with a *renaming change* delta.
- If a complex element retains the same name and the same structure, but a non-identifying attribute has changed, it is marked with a *change* delta.
- If an element does not exist in the newer version, it is marked with a *deletion* delta.
- If a new element does not exist in the older version, it is marked with an*addition* delta.
- If an element is mapped between the two versions but has a different parent element, it is marked with a *move* delta.
- If an element is mapped between the two versions but has a different parent element and one or more its non-identifying attributes was changed, it is marked with a *move and change* delta.

The above rules have been described in depth in our previous work [5]. The comparison process produces a diff script, which follows the structure of the original service interface, with each interface element annotated by its corresponding delta.

The **adaptation** step currently deals with Java clients only. The adaptation algorithm as described in our previous work [6] is language independent and can be implemented for any language provided that it has a way to manipulate the source code like the Eclipse JDT[5] for Java. Because Java client proxies are generated based on the service interface, the adaptation process, guided by the service-interface deltas, modifies the old-service proxy to invoke the new service interface, thus eliminating the need to modify the actual client. The tool currently works for client proxies generated by Apache Axis2[6] from the WSDL interface[7]. The adaptation tool injects in the old proxy code fragments that, for each operation, (a) copy the input provided client into an instance of the input type required by the new operation version; (b) invoke the new operation; and (c) repackage the returned output values to the output types of the old operation version, as expected by the client application. In effect, the new code fragments implement a middleman translation process, enabling the client application to interact with the new interface through the old-interface protocol. This effectively makes the service-interface changes

[5]http://www.eclipse.org/jdt/
[6]http://axis.apache.org/axis2/java/core/
[7]Currently, client adaptation is only available for WSDL services.

transparent to the client code. In this way, it is ensured that no actual changes are applied to the client code but rather to the auto-generated client proxy and, thus, the awareness of the client developers of their code is not affected. The adaptation process is described in more details in our previous work [6]

Finally, the **regression testing** tool launches the JUnit test cases of the client application and reports the results in the Eclipse JUnit view. It is assumed that the developer has already specified a run configuration of the test cases in the current workspace, simply by running the tests once manually. Successful tests indicate that there was not any change in the functionality of the service but just in its interface. In case of failed tests, JUnit points directly to the operation that failed. Furthermore, the information provided by the diff script exactly shows what was changed in this particular operation. Therefore, the developer can go to the exact place in the source code and make the necessary changes manually.

In summary, in the best case scenario, where the service interface has changed but its behaviour has not, WSDarwin automatically adapts the client application. In the worst case scenario, where the service behaviour has also changed, WSDarwin adapts the client and provides the client-application developer with information relevant to completing the additional changes required, thus systematizing and decreasing the development effort necessary.

A. User Interaction with WSDarwin

WSDarwin is implemented as an Eclipse plug-in with a single-view type of interface. Figure 1 shows the WSDarwin user interface within the Eclipse workspace. The plug-in contributes a new menu in Eclipse's menu bar with two items (Figure 1(1)). The first item launches a view for WSDL interfaces, while the second launches a view for WADL interfaces. The two views are equivalent in appearance and functionality.

Comparison: The view has three buttons (Figure 1(2)), each one corresponding to one of the WSDarwin tools. When the comparison button is selected, the user is prompted to provide two versions of the service interface and the corresponding versions of the client proxy. Then, the tool calculates the differences and the view's table is populated with the deltas (Figures 1(3) and 2). The first column of the table contains the type of the delta, the second contains the type of the service element, the third contains the identifier of the source (old version) element, the fourth contains the identifier of the target (new version) element and the remaining columns contain additional information about the attributes changed in case of a change delta. The results are presented in a tree format to correspond to the service interface's structure.

Adaptation: The adaptation functionality of WSDarwin treats the whole process as a refactoring. Using the Eclipse Language Toolkit (LTK)[8] every change is recorded. When the adaptation button is selected, a preview is shown to the user with all the changes that will happen to the client proxy (Figure 3). After the preview has been accepted by the user, the tool performs the actual changes on the AST level using the Eclipse Java Development Toolkit (JDT)[9] and opens the updated client proxy in the Eclipse Java editor (Figure 1(4)). Due to every change being recorded using LTK, the whole process is reversible and can be undone from the Eclipse interface.

JUnit testing: Finally, the "run tests" button searches in the client project for test cases and invokes the Eclipse JUnit shortcut to execute them. The results are then presented by the JUnit view of the Eclipse interface (Figure 1(5)).

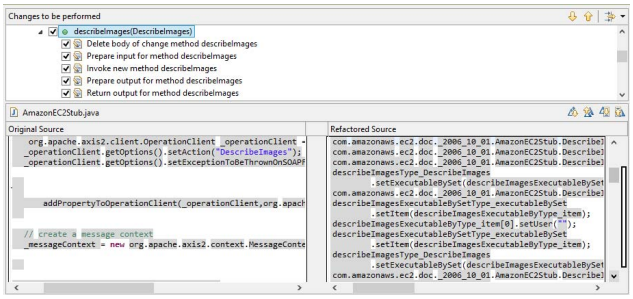


Fig. 3. The refactoring preview window for the adaptation of the client proxy.

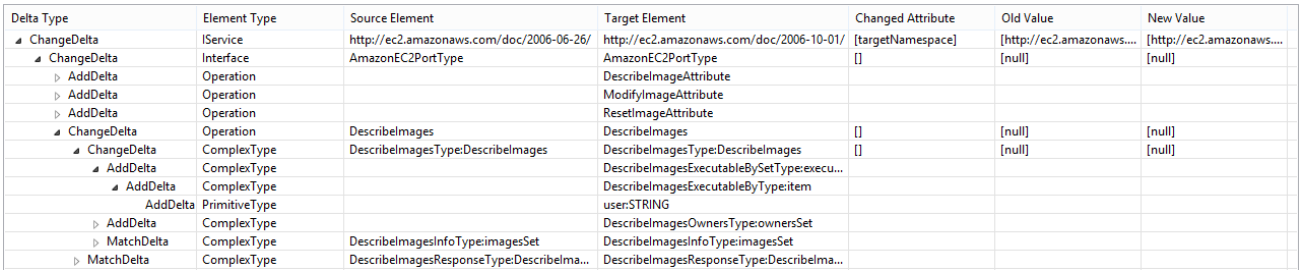
III. Client-Adaptation Case Study

We applied the WSDarwin tools on a real-world web service. The example web service is the Amazon EC2 web service to manage the Amazon Elastic Cloud infrastructures. We applied WSDarwin on 25 versions of the service and measured the time for each individual component as reported in Table I. The results were obtained with a machine that has an Intel Core i7-3517U processor at 2.4 GHz, 8GB RAM and 64-bit operating system. We calculated only the time that was required for the tools to complete each task and not the time required to manually provide input or make choices. The times were measured once for each version.

As it can be seen from the Table, the performance of the tool depends on the size of the interface, but even in the latest versions for more than 2000 elements, the whole process didn't take more than 5 minutes. The parsing of the service interface takes from less than a second to little more than a minute for the later versions. During the parsing process, the WSDL interface is parsed using a DOM parser and it is then transformed into a WSMeta representation which will be passed to the comparison module. The performance of parsing is achieved by using an efficient way to resolve references within the service interface; the first time the WSDL interface is parsed with the DOM parser, its constituent elements are stored into `Map` structures, where the key is the identifier of the element and the value is the element itself. This way, when a reference needs to be resolved, it simply queries the Map and gets the XML node for further processing without having to traverse the whole structure. This results into a linear complexity.

Having resolved the references during parsing, the comparison of the service interfaces becomes extremely efficient, taking merely milliseconds, as it can be seen from the Table. This is achieved because WSMeta strips the interfaces of some unnecessary elements that only serve as references imposed by the underlying XML constraints. Furthermore, since the

[8]http://www.eclipse.org/articles/Article-LTK/ltk.html
[9]http://www.eclipse.org/jdt/



Delta Type	Element Type	Source Element	Target Element	Changed Attribute	Old Value	New Value
ChangeDelta	IService	http://ec2.amazonaws.com/doc/2006-06-26/	http://ec2.amazonaws.com/doc/2006-10-01/	[targetNamespace]	[http://ec2.amazonaws....	[http://ec2.amazonaws....
ChangeDelta	Interface	AmazonEC2PortType	AmazonEC2PortType	[]	[null]	[null]
AddDelta	Operation		DescribeImageAttribute			
AddDelta	Operation		ModifyImageAttribute			
AddDelta	Operation		ResetImageAttribute			
ChangeDelta	Operation	DescribeImages	DescribeImages	[]	[null]	[null]
ChangeDelta	ComplexType	DescribeImagesType:DescribeImages	DescribeImagesType:DescribeImages	[]	[null]	[null]
AddDelta	ComplexType		DescribeImagesExecutableBySetType:execu...			
AddDelta	ComplexType		DescribeImagesExecutableByType:item			
AddDelta	PrimitiveType		user:STRING			
AddDelta	ComplexType		DescribeImagesOwnersType:ownersSet			
MatchDelta	ComplexType	DescribeImagesInfoType:imagesSet	DescribeImagesInfoType:imagesSet			
MatchDelta	ComplexType	DescribeImagesResponseType:DescribeIma...	DescribeImagesResponseType:DescribeIma...			

Fig. 2. Part of the diff script for the Calculator service.

references are already resolved, the elements are compared as they are being traversed once.

Finally, the adaptation is the heaviest process of the three since it involves complex AST transformations. The most intensive task is finding the corresponding Java class of the client proxy from a WSDL complex type. Axis2 generates a Java class for each complex type in the service interface, from which we need to get all the getters and setters to adapt the client. The adaptation begins by traversing all the classes generated for the client proxy and storing them in a `Map` structure, where the key class-identifier key is the same as the identifier of the corresponding complex type.

TABLE I. THE EVOLUTION OF THE AMAZON EC2 WSDL INTERFACE.

Version	Operations	Types	Parsing time (ms)	Diff time (ms)	Adaptation time (ms)
2006-06-26	14	167	2735		
2006-10-01	17	189	414	19	3078
2007-01-03	19	201	374	12	4636
2007-01-19	19	204	356	16	4461
2007-03-01	20	219	437	15	5053
2007-08-29	20	222	467	32	4988
2008-02-01	26	276	681	24	5796
2008-05-05	34	356	907	21	7947
2008-08-08	37	468	1264	18	12268
2008-12-01	38	479	1709	30	19171
2009-03-01	41	521	2176	30	16065
2009-04-04	43	549	2739	27	22406
2009-07-15	65	810	4479	48	19999
2009-08-15	68	841	7321	56	38527
2009-10-31	74	936	9223	336	48571
2009-11-30	81	1099	12181	35	61459
2010-06-15	87	1166	14516	6	78317
2010-08-31	91	1399	26302	8	87606
2010-11-15	95	1467	29284	5	93864
2011-01-01	118	1835	49269	33	92388
2011-02-28	118	1846	49511	19	177255
2011-11-01	119	1907	51907	18	140332
2011-12-01	127	2121	63872	555	158333
2011-12-15	128	2147	60842	27	200398
2012-03-01	132	2203	70462	32	190552

IV. CONCLUSIONS

In this paper, we described the WSDarwin tool, an Eclipse plug-in to support web-service evolution. We have recognized the need to allow developers of client applications to react to the evolution of the services they consume by employing only publicly available information. WSDarwin offers a complete solution to this problem. First, it is able to compare two versions of the service interface and allow the developer to study the evolution of the service. The results of the comparison are then provided as an input to the adaptation process. WSDarwin employs a methodology that would adapt the client application without actually altering the manually developed code. It takes advantage of the fact that WSDL-based services can be invoked through auto-generated client proxies. It transforms the proxy corresponding to the old version of the service to invoke the new proxy. Eventually, the client code is still aware of the old interface but it now gets the new content from the updated service, thus making the changes transparent to the consuming application. Finally, the user can automatically run JUnit test cases for the client application through WSDarwin, either to confirm that its function was not disrupted or to gather more information about any additional manual changes that might be necessary.

We plan to further improve the WADL module of WSDarwin. At the current version of the tool, only the comparison of WADL interfaces is available, which is also subject to similar limitations as with the WSDL interfaces (i.e., different generation mechanisms). For the adaptation, we will rely on similar client proxy generation tools like the Apache Axis2. More specifically, we can use the `wadl2java` tools provided either by Oracle[10] or by Apache[11]. The two tools roughly produce the same client proxy. We can apply the same adaptation algorithm as in the case of WSDL interfaces, but paying extra attention on how to handle such REST concepts as resources and URIs.

REFERENCES

[1] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani, "Developing adapters for web services integration," in *CAiSE*, 2005, pp. 415–429.

[2] S. R. Ponnekanti and A. Fox, "Interoperability among independently evolving web services," in *Middleware '04*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 331–351.

[3] P. Kaminski, M. Litoiu, and H. Müller, "A design technique for evolving web services," in *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research - CASCON '06*. New York, New York, USA: ACM Press, Oct. 2006, p. 23.

[4] M. Fokaefs and E. Stroulia, "WSMeta: a meta-model for web services to compare service interfaces," in *Proceedings of the 17th Panhellenic Conference on Informatics*. ACM, 2013, pp. 1–8.

[5] ——, "WSDarwin: Studying the Evolution of Web Service Systems," in *Advanced Web Services*. Springer New York, 2014, pp. 199–223.

[6] ——, "WSDarwin: automatic web service client adaptation." in *CASCON*, 2012, pp. 176–191.

[10] https://wadl.java.net/wadl2java.html

[11] http://cxf.apache.org/docs/jaxrs-services-description.html#JAXRSServicesDescription-wadl2javacommandlinetool

Chapter 3

Support for REST Applications

REST services are typically developed in a more ad hoc and less structured way than WS-* services. They are not always supported by standard specifications; the interface of a REST API is usually published as a free-text, semi-structured HTML page. This non-standardized way of publishing a web service sometimes gives rise to variability on how the functionality of the service can be invoked by clients. This lack of standardization deprives client developers from systematic support, necessary for the development and maintenance of their applications. However, REST APIs are grounded on the popularity of the HTTP protocol and the ease with which such requests can be constructed. All programming languages provide support to construct HTTP requests and to analyse XML and JSON responses, which are the typical formats for REST services. The lack of systematicity is a double-edged sword; on one hand, it makes adoption of REST APIs easy since there exists programmatic support for handling the protocols and formats used by the REST style, but on the other hand, it makes handling the evolution of a REST service difficult, since there is no systematic way to find how the service changed and how the change can be dealt with. In other words, the lack of a machine-readable interface significantly lowers the ability to create automated tools and support the development and maintenance of REST applications. Although it has not been widely adopted, W3C has proposed a structured format, similar to WSDL, to publish REST APIs, called Web Application Description Language (WADL). The WADL file specifies the structure of service's resources, the methods that

are available for each resource and the methods' input and output variables. Furthermore, there are tools, similar to those for WS-* services, to automatically create client proxies from WADL interfaces.

Given that WADL interfaces will offer more opportunities for automation but, at the same time, given the reluctance of developers to adopt such a structured interface, I developed an automatic **service-interface specification generator** for WADL interfaces of REST services. The tool is designed to be easy to use and requiring little input, so that it doesn't add overhead to the development process. The tool requires as input a set of URL requests for the REST service, which can be assumed that are already constructed to either invoke the service or test the client application. The requests are then analysed to infer the resources of the service and the input data for the methods that are available on these resources. The same requests are used to invoke the service and the responses are analysed to infer the schema of the output data. The analysis results for each request are merged in order to produce a single WADL for the REST service. Batch analysis of requests gives the ability to resolve parameter types with higher confidence, identify enumerations, identify resources with variable identifiers and create a more complete interface for the service. The tool can be used both by providers and clients, which implies that it releases any particular party from the responsibility of creating or having to provide a WADL interface.

An efficient method to generate WADL interfaces can facilitate and automate a variety of tasks, including the generation of client proxies for any programming language, the study of the evolution of a REST API, service discovery and service client adaptation. I demonstrate the use of the WADL generator on the Tumblr API and discuss how the generated WADL interfaces can be used for the aforementioned tasks focusing more on the client proxy generation and service evolution. Given the opportunity to demonstrate the WADL generator, I also study the practices on developing and maintaining REST applications, using Tumblr as a case study. The study reveals that there are discrepancies on how REST principles are perceived and implemented by developers even between different versions of the same API. The study also

shows that best practices are not always followed. More specifically, Tumblr evolved radically between versions 1 and 2, its documentation, especially for version 1, is rather unstructured and disorganized and the developers of the API have manually developed SDKs on a small set of programming languages for client applications. All these are tasks that can benefit from the existence of a formal service interface, like a WADL file.

To further support the development and maintenance of REST applications, I propose a **cross-vendor service mapper** to map the output elements between two similar services from the same domain, but offered by different providers. The premise of the method is that two parameters from the different services that contain the same data correspond semantically to the same entity. Therefore, the developer will know what data can be used to invoke another similar service. The assumption that similar data can exist between two services can be valid in a variety of ways, either within the same domain (e.g. movies, sports), or temporally constrained data (e.g. ticket selling services), or geographically constrained data (e.g. weather data or map directions). The goal of this method is to allow client developers to discover similar services and then easily migrate their application between these services. In order to evaluate the method, I apply it on services from two domains (movies and maps). The evaluation shows that the method is fairly accurate, especially concerning services that are particularly popular (e.g. IMDb for movies or Google Maps for map). It is revealed that other providers tend to make an effort to be compatible with the leading service of a domain, probably to facilitate the migration of client applications from the leading service to their services. The need and usefulness of the mapper as a tool to support service migration is accentuated by the fact it improves effort (in terms of time) by 6 or 7 times. In spite of being a semi-automatic method and not fully automatic, the manual steps are only needed to map the input parameters of two services, which are much less in number and of lower complexity than the output parameters, which are automatically mapped between the two services. The method can be complemented structural and semantic comparison techniques to further improve its results and its efficiency.

The **service-interface specification generator** for WADL interfaces, a **service-interface comparator**, similar to the one proposed for WSDL interfaces, a **client-proxy generator** to generate Java-based service client proxies from WADL interfaces and the **cross-vendor service mapper** are implemented in a web application. The web application was developed in the spirit of the general REST principles, in the sense, that it is simple to use, requires minimal input and it can efficiently produced the necessary artifacts and information to promote automation and facilitate a variety of development tasks. For the interface generation, the web application presents an editable version of the generated WADL file, where the user manually edit names and types of parameters, methods and resources and add or remove elements from the file. After the file is edited, it can be downloaded locally. For the comparison, the web application takes advantage of the fact that WSDarwin can also express WADL interfaces in WSMeta and uses the same comparison method as in the case of WS-* services. The two versions are then presented side by side with the changes annotated with the appropriate colors. Finally, for the cross-vendor comparison, the web application shows what output elements from one service are mapped to the other service and how similar their values are. The elements can be selected and are highlighted on the two WADL interfaces which are shown side by side on the right of the screen.

3.1 Developing and Maintaining REST Client Applications: The Tumblr Case Study

Fokaefs, M., Stroulia, E., 2015. Developing and Maintaining REST Client Applications: The Tumblr Case Study. In: IEEE International Conference on Software Engineering, Software Engineering In Practice (ICSE 2015 SEIP). IEEE.

Note: This paper has been submitted to the Software Engineering in Practice (SEIP) track of the International Conference on Software Engineering (ICSE 2015) pending review.

Developing and Maintaining REST Client Applications: The Tumblr Case Study

Marios Fokaefs and Eleni Stroulia
Department of Computing Science
University of Alberta, Edmonton, Alberta, Canada
Email: {fokaefs,stroulia}@ualberta.ca

***Abstract*—Service orientation has been established as the dominant paradigm for the development of modular distributed software systems. In spite of the substantial research effort dedicated to the development of methods and tools to support SOAP-based service-oriented application development, in practice, RESTful services have surpassed SOAP-based services in popularity and adoption, primarily due to the simplicity of their invocation. However, poor adoption of REST specification standards and lack of systematic development tools have given rise to many, more or less compliant, variants of the RESTful style constraints, which undermine the evolvability and interoperability of these systems. In this paper, we reflect on the principles and practice of the REST style, we describe a tool that we have developed to encourage the standardization and systematization of RESTful application development, and we illustrate the usefulness of our tool in the context of the Tumblr evolution.**

I. Introduction

Service-oriented systems have been established as the dominant paradigm for the development of modular software. Due to attributes such as dynamic binding, information hiding, and invocation through HTTP-based protocols, web services offer a great degree of flexibility and efficiency, which are considered as assets in the business of modern software systems.

Within the domain of service technologies, REST services have recently gained much popularity as the prevalent architectural style to develop service-oriented systems. This popularity can be attributed to the fact that they can be invoked through simple HTTP requests, which makes them even more easy to use since practically every programming language provides software and tooling to support HTTP requests. The increasing popularity of REST services, especially over other technologies like SOAP services, is also evident in the recent developments in the software industry. Big companies like Amazon [1] and Google [2] have discontinued their SOAP APIs in favour of their RESTful counterparts. Amazon has also reported that 80% of the requests to their services comes through their REST API [3] and that querying the services using REST is 6 times faster than with SOAP [4].

Software-engineering research has already contributed numerous methods to supporting the development and maintenance of service systems. There exist many automatic and interactive tools to support service discovery, selection and binding, web service maintenance and evolution, web-service middlewares and more. This work has focused primarily on SOAP-based services. The relative scarcity of tool support for engineering REST-based systems is due to the fact that the simplicity of REST services deprives researchers from some very valuable artifacts. For example, the interfaces of REST services are usually published as semi-structured HTML pages that do not follow the same standards across different providers and are specified in free text. Furthermore, these specifications may be incomplete; for example, responses are usually specified through simple and incomplete examples. All these factors may hinder some software-engineering tasks like service discovery or addressing service evolution. Although there exist standard interface formats like WADL [5] or WSDL 2.0 [6] to specify REST services, providers keep publishing REST APIs as HTML pages, which are easily understandable by humans but not as easily consumable and reasoned about by software.

A challenge even more important than the informality of their public documentation is the high degree of variance in their design and evolution. Variation in the conventions of designing a REST API can occur between different vendors or even within a single API. For example, with respect to the type of the response (e.g. JSON or XML), there are services that return only one type (e.g. Twitter [7] or Tumblr [8]) or those that return more than one (e.g. Google [9] or RottenTomatoes [10]). The type may be specified as an extension to the URL request (Twitter and RottenTomatoes) or as a separate resource or parameter (Google and Tumblr). Another example of variability that occurs within the Twitter API, where in some requests the identifier to a specific resource is specified as a separate resource (variable id), while in others as a parameter. These differences can cause problems to consumers of such services and they can make the maintenance of client applications difficult and costly and hinder the development of automated tools consistent across all domains and providers. One such problem is that consumers may become tightly coupled with their providers (similar to the "vendor lock-in" problem in cloud computing [11]), which is against the general flexibility principle of REST architectures. Under the assumption that best practices cannot always be followed, in this work, we argue that automatic tools, systematic methods and standardized development can alleviate some of these problematic situations and assist the developers of REST client applications.

In this work, we present an overview of the variability points

of REST services and their typical implementations by various providers. We pay particular attention to issues concerning the evolution of REST services and the maintenance of client applications. We argue that the first step towards standardizing the development of REST applications and facilitating their maintenance and evolution with automated tools is the construction of standard WADL interfaces for REST services in an automatic and efficient manner causing no additional overhead to either providers or= clients. To this end, we have developed such a tool and present it as an extension for the *WSDarwin* toolkit for service client adaptation [12].

We use the Tumblr API [8] as a case study. We discuss the various design and evolution decisions throughout the development of a REST API like Tumblr, and consider their implications to potential client applications. Furthermore, we evaluate the *WSDarwin* WADL generator on the Tumblr case study and demonstrate its usefulness to client-applications developers.

The rest of the paper is organized as follows. In Section II, we discuss the REST architectural style, its technical details and how these may be implemented in real world with particular interest in automatic methods. Section III introduces *WSDarwin*'s contribution to set of tools to support the development of REST applications, an automatic WADL generation tool. In Section IV, we present the case of the Tumblr API, on which we studied the application of the REST principles and techniques and we demonstrated the application of the WADL generator on a real example. In Section V, we present a broad overview of research literature related to the evolution and maintenance of service systems focusing more specifically on REST applications. Finally, Section VI concludes our work and briefly discusses some of our future plans.

II. REST in Theory and Practice

The REST (Representation Stateless Transfer) protocol and architectural style was first described by Roy Fielding [13], who identified a set of six architectural constraints governing the design of an emerging class of web applications at the time. At the same time, the web-services (WS-*) stack of standards was being proposed as a complete systematic solution for specifying, engineering and maintaining service oriented systems. The two approaches were in effect motivated by the same needs, yet they were fundamentally different conceptually. RESTful systems, as described by Fielding, operated on a "lighter" medium, meaning that there exists no intelligent middleware as is the case in WS-* services and the handling of the constraints lies on the communicating participants, effectively decoupling the client from the server.

In practice, the REST architecture style, lacking any standardization and reference implementation, was interpreted differently by various providers, who practically started offering simple web services over HTTP without much concern regarding the REST architectural guidelines. House [14] provides an overview of the most common "violations" of the REST principles by service providers. The usual suspects include non-conformance with HTTP principles, e.g. how to properly invoke the HTTP methods and use the HTTP codes, and reduction of the flexibility and dynamic nature of REST APIs, which makes them harder to be discovered.

In the context of our work, we are more interested in the conventions that may affect service clients, so we focus on the interface of the service and how it evolves. The interface of a REST service, or more specifically for a single request, consists of the service endpoint, the path to access the desired resource, and any parameters of the request. The service endpoint specifies the address of the server on which the service is deployed. A frequent convention that may affect the endpoint is that providers specify the version of the service as part of the endpoint. Therefore, when a new service version becomes available, the endpoint also changes and any clients accessing the service through the old endpoint break. An alternative to this convention is to specify the version of the service as a parameter to the request, which, if omitted, it will default to the latest version. Nevertheless, the need to specify the version of the service stems from the practice of maintaining multiple versions, which is, to being with, a costly process for the provider [15], [16].

A second convention that affects the interface of a service is how the provider decides to specify the service *resources*. The resource is the basic abstraction of information in the REST architectural style; and, in essence, any concept can be a resource. In practice, we can consider that the data of a service is organized as a relational database, where the resources are the tables and the HTTP requests correspond to SQL queries. The user of the database can access whole collections (tables) or query for specific records. If the tables are indexed, then the user can access specific records simply by using the respective key. Otherwise, the query needs to specify particular constraints to access the desired records. As these relational repositories become exposed through REST services, the question becomes to specify what should be considered as a resource. A specific instance of a resource can be specified as part of the URL of the request or as a parameter. If this convention is changed from one version of the service to another, this will break the consuming applications. REST also gives the option to specify resources with variable identifiers, referring to a class of resources (e.g. accounts, usernames). The identifier of the actual resource and, thus, the complete URL of the request is determined in runtime by the client of the service.

A. Standardized REST Services Specifications

As we have already discussed, REST services are usually documented with semi-structured HTML pages that define how the URL requests are constructed and the structure of the responses. However, these pages are not machine readable which hinders the applicability of automatic software engineering tools.

To address this shortcoming, two standardized formats for specifying REST services have been put forward, *i.e.,* WADL [5] and WSDL 2.0 [6], which, although not very popular, can be consumed by tools. WSDL 2.0, as a successor

of WSDL, is more tailored to the structure of the operation-oriented SOAP web services. WADL, a W3C submission, is targeted particularly to REST services. As such, it describes the structure of the data that becomes available through the service as resources and the HTTP methods that can be applied on these resources.

The WADL specification of a REST service interface describes the service endpoint, a collection of available resources each one associated with its corresponding path, and the methods it exposes. For each method, WADL specifies its input and output parameters, typically as *representation*, which is a reference to particular data structure defined in the service's XML schema. WADL allows for the definition of resources with variable identifiers (surrounded by curly brackets), in which case the variable ID is also defined as a parameter of the resource. Furthermore, in case of direct access of a resource (where the identifier of the resource is part of the URL request), the method corresponding to the request does not have an identifier and only the HTTP method is specified (`GET`, `PUT`, `POST`, `DELETE`). More details about the structure of a WADL interface can be found in [5] and an example will be provided for the Tumblr API in Section IV.

B. REST Client Generation

The main motivation behind web services is to increase software interoperability by decoupling clients and providers and making their dependency technologically agnostic so that a client may invoke a service, regardless of its programming-language implementation and platform. This can be achieved by special-purpose middleware [17], [18] that consumes the machine-readable service-interface specification and produces client proxies in a variety of programming languages. IN this manner, developers can obtain a service client in the programming language of their choice, from the single service interface specification.

Although such tooling does exist, due to the low adoption of standards, providers tend to offer client-development toolkits in a variety of languages to facilitate consumers in invoking the service through a native interface. While this practice eliminates the need for a special middleware, it increases the coupling between the client and the service provider, violating the intent behind service orientation. First, this method implies the need for a toolkit for every programming language that exists currently or may exist in the future, which is a requirement unlikely to be met. Second, if there is a client that uses a toolkit for a particular language and they later decide to migrate to another language, this will transform the particular piece of client code into legacy software, a problem that web services set out to solve in the first place. Finally, dedicated client-development toolkits reduce the interoperability of client applications. If a client is tightly coupled with a toolkit and the developers decide to migrate to another service, the transition is bound to be harder than it would have been if the client was developed from the service interfaces using the same middleware under the same assumptions. The consistency of the code produced by a single middleware for a variety of web services can be a great advantage for clients that wish to switch between service providers.

C. REST Service Evolution

In our previous work [19], we have explored several evolution scenarios that may occur on SOAP web services and classified them according to their potential effect on client applications as *no-effect*, *adaptable* and *non-adaptable*. A no-effect change will not affect existing clients, adaptable changes can be addressed by automatic tools and non-adaptable changes need extra manual effort to be addressed. In this paper, we explore similar cases for REST services.

TABLE I
EVOLUTION SCENARIOS IN REST SERVICES.

Change	Impact	Condition/ Note
Endpoint Change	Adaptable	The endpoints can automatically be mapped.
	Non-adaptable	The endpoints cannot be mapped and the service cannot be resolved.
Resource Addition	No-effect	The resource is added to the end and is not used.
	Adaptable	The position of the new resource can be automatically identified and the new resource can be automatically injected to existing requests.
Resource Renaming	Adaptable	The resource can be mapped based on its structure.
	Non-adaptable	The structure of the resource has also changed.
Parameter Addition	Non-adaptable	The new parameter (input) is required and its value cannot be resolved solely from the interface.
	No-effect	The new parameter (input/output) is optional.
Input Parameters Reordering and Type Change	No-effect	Request parameters are untyped and the order is irrelevant.
Output Parameters Type Change	Non-adaptable	The client casts variable or parses strings to numbers without type checking.
Output Parameters Reordering	No-effect	Output parameters are usually wrapped in complex types and their order plays no role.

Table I summarizes a collection of evolution scenarios for REST services, along with their classification based on their impact on clients. Some of these scenarios are further discussed as they have appeared in the Tumblr case study (see Section IV). The particular changes to a REST service that can potentially affect client applications are those around the formation of the URL requests towards the service, namely changes to the service endpoint, the structure of the resources and the parameters of the methods. Changes to the response of a service may have negative effect on the client if they are not handled properly by the application. However, new output parameters will not affect existing clients, since they will be unaware of them.

Changes to the service endpoint will affect all the URL requests constructed for this service; consequently, current clients issuing these requests will break because they will no longer be able to resolve the service address. However, if the new endpoint can be easily mapped to the old endpoint,

then the URLs can be systematically (even automatically) changed to reflect the new endpoint. The mapping can be achieved semantically, if the terms comprising the endpoint are somehow related to each other, or structurally, if the contained resources and their methods are the same or similar. If, however, there were extensive changes to the rest of the service-interface structure, then it would become more difficult to map the two versions of the endpoint, especially if the service contains more than one endpoints. In that case, the client would require additional information about the new service version, and more manual effort would be necessary to adapt to the change.

Changes to the structure of the service resources may cause similar but less detrimental problems. For example, if a new resource is added to the service and that resource is independent from the existing ones, no former URLs will be affected. However, if the organization and the taxonomy of the resources is changed and the path for a new resource is added in a position that affects the structure of existing URLs, this will break clients issuing these requests. For a simple example, let us consider a service for biological data, which have resources about plants and animals, and under those it defines resources for specific species, e.g. pine trees, rose bushes, dogs, mice and so on. Let us assume that in the second version, some new classes are considered that are more specific than plants and animals but more generic than the species, e.g. trees, bushes, mammals, birds and so on. In this case, existing requests will have to be changed in order to add the intermediate resources. Renaming or changing the structure (subresources, methods) of existing resources may have the same breaking results for clients, but as long as the resources can be mapped between the two versions, the changes are adaptable by clients applications, since they are already in possession of the required data to reissue the requests in the new format. We consider two elements (resources, parameters or methods) to be mapped between two versions if they retain their names or their attributes (or both) or carry the same information in both versions.

The impact of changes in the parameters of REST methods depends on the importance of these parameters. If a new required parameter is added to an existing method, current client requests will break. If the parameter is optional, or has a default value, then current requests will still be valid in the new version. An interesting difference between REST and SOAP services is that, in REST, parameters types are not explicitly declared, but inferred from their values and appropriately handled by the service. From the client's perspective, all parameters are specified as alphanumerics and are passed to the service. Therefore, changes to the type of a parameter are of no effect to the client. Finally, unlike other programming styles, the order of parameters in REST requests plays no role whatsoever. If the type of the output parameters is changed, then the impact on the client will depend on how the consuming application handles the service response data. For example, if the client expected a particular type and cast the corresponding variable without first checking the returned type, then the application would break, if the type of the returned parameter changed.

III. The *WSDarwin* REST Toolkit

WSDarwin [12] offers a collection of tools to support the evolution of web services and the adaptation of service clients, complying with web-services principles and aware of actual development practices. *WSDarwin* operates under the assumptions that providers and clients share the minimum required information regarding the implementation of their software, *i.e.,* services and applications, and that evolution decisions are made independently, without consultation between providers and clients. As a result, service clients may have to deal with unexpected breaking changes to web services with little information. *WSDarwin* offers the ability to analyse software, generate middleware assets, and manipulate client applications using only publicly available software artifacts in a systematic and interactive manner.

In order to support the development and evolution of REST applications, we extend the *WSDarwin* toolkit to enable the automatic generation of WADL interfaces for REST services. This extension is built in accordance to the general philosophy of REST services; it is lightweight and easy to use, and it operates under the realistic assumption that some best practices may not necessarily followed. We have made the methodological commitment to reflect and accommodate the practical realities of RESTful development rather than dictating best practices. Its simple interface and degree of automation is conceived to make it a useful tool for service developers and client developers alike. Service developers can use the application to provide additional information to their clients about the application in an efficient and cost-effective manner. Service clients can obtain more information about a service even when this information is not provided by the service developers.

A. WADL Generation

WSDarwin can produce the WADL specification for a REST service simply by exercising the service and analysing its requests and responses. The availability of an automated tool for generating a well-define service-interface specification on demand relieves the provider from the responsibility of constructing this specification beforehand. Second, it allows service clients to use this interface to generate client proxies, compare similar alternative services or service versions, and, in general, take advantage of the software-engineering tools available for web service development.

In order to generate the WADL interface, *WSDarwin* requires as input one or more URLs corresponding to service requests, at least one request per method. Clearly, the more URLs are given to *WSDarwin*, the more complete the interface specification that it will generate. During the batch analysis of multiple requests, the intermediate results are merged to avoid duplicates and eventually a single WADL file is produced for the service under examination. The generated file is valid as per the WADL schema provided by W3C [5].

The input URLs are analysed using the rules provided by the RestDescribe tool [20], which is a similar tool to generate WADL interfaces. Figure 1 shows how a request is parsed according to these rules. The first part (authority) corresponds to the service endpoint, *i.e.,* the address of the service. In the WADL, the endpoint is the `base` attribute of the `resources` element. The next part of the URL (path) specifies the individual resources, whose paths are separated by a slash ("/"). The individual resources are organized hierarchically in the WADL file and each resource is nested within the previous one.

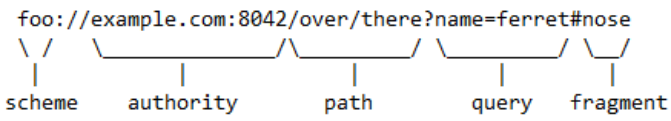


Fig. 1. The analysis of an input URL in its components

Each URL request is a method that is added to the last resource specified by the URL. The HTTP operation of the method, *i.e.,* GET, PUT, POST, DELETE, is not part of the URL and has to be specified explicitly by the user. If the resources in the URL are followed by a question mark ("?"), this means that the method has parameters, which come in name-value pairs, separated with an ampersand ("&"). These parameters, along with their inferred types (the type-resolution heuristic method is describe din detail in Section III-A1), are added to the `request` part of the method. If the URL has no parameters, in which case the URL tries to access a specific resource or a collection of resources, the method does not have a request part.

Once the requests have been analysed, they are used to invoke the service and obtain the corresponding responses, which are returned either as XML or as JSON. *WSDarwin* automatically determines the format of the response, since this is not always explicitly specified, which is then specified as the `mediaType` of the response. The response is then parsed and analysed to infer the underlying schema. The identified types of the output parameters and their structure are added as an XML schema in the `grammars` element of the WADL file. A `representation` element is added in the `response` part of the WADL method with a reference to the corresponding type in the grammars part.

The generated WADL file becomes available to the developer, who can further edit it by changing the attributes of an element, or adding and removing elements, or editing attribute types.

A significant feature of *WSDarwin* is its ability to batch process multiple requests for a service. This enables the tool to resolve parameter types with high confidence, to identify enumeration types, and to identify resources with variable identifiers. The assumption that the user of the tool will be easily able to provide multiple URLs is quite realistic. To begin with, developers can use the URLs mentioned in the service HTML documentation. Furthermore, they are likely to have further request URLs as part of their application testing.

A second important *WSDarwin*feature of *WSDarwin*'s on-demand WADL generation is that the produced WADL specifies only the parts of the service used by the client application. This results in a concise and compact interface without unnecessary data, a fact that can facilitate the maintenance of the client application. While this may somewhat limit the extendibility of the client application (a service method not currently used might still become useful in the future), the WADL-generation process is so simple that it can be invoked at any point in time to produce a new WADL to meet the current client-application requirements.

1) Type Resolution: Given that REST requests and responses do not specify types for their parameters, the identification of a parameter's type by analysing its value is a critical part of the WADL generation. WADL specifications must include type information because certain programming languages, for which client proxies can be generated, explicitly require them.

WSDarwin can identify a variety of primitive types as specified in the W3C definition for the XML Schema [1]: `string`, `double`, `float`, `long`, `int`, `short`, `byte`, `dateTime`, `date`, `boolean` and `anyURI`. Each type is determined based on a regular expression and, if necessary, by a set of specific conditions (Table II). Given the value of a parameter, the expressions are checked from the most specific to the most general type. In essence, everything can be expressed as a string and every number can be expressed as a double. So, these general types are checked last. Numeric types differ from each other based on the specific range of numbers they cover. Therefore, numeric values are subjected to additional conditions to check for the particular range they belong to. If a parameter is a list of values, then a complex type is created for the list and XSD element is added to the type for the contents of the list. The type of the element is determined as previously and an additional attribute is added (`maxoccurs=''unbounded''`).

When processing multiple URLs, if a parameter is found to have more than one different types for different requests, then the final type is the most generic type of all the identified candidate types (e.g., `string` is more generic than `int`) under the assumption that the generic type can subsume the values of all the more specific types. When presenting the final WADL file, the tool also presents its level of confidence for each identified type, where confidence is measured as the percentage of the total requests processed. For example, if a parameter was found to be string in 7 out of 10 processed URLs, then it is reported as string with 70% confidence.

a) Identifying Enumerations: Enumerations are a special type, whose values are restricted within a predefined set. In a WADL file, enumerations can be implemented in one of two ways, either as `option` elements of a `param` or as `simpleType` in the XML schema with a `restriction` element that contains the possible values as `enumeration` elements. The *WSDarwin* WADL generator adds `param`

[1]http://www.w3.org/TR/xmlschema-2/

TABLE II
REGULAR EXPRESSIONS TO IDENTIFY PARAMETER TYPES.

Type	Regex
double	`^[-+]?[0-9]+[.]?[0-9]* ([eE][-+]?[0-9]+)?$` AND `parseDouble == TRUE`
float	`^[-+]?[0-9]+[.]?[0-9]* ([eE][-+]?[0-9]+)?$` AND `parseFloat == TRUE`
long	`^[-+]?\\d*$` AND `parseLong == TRUE`
int	`^[-+]?\\d*$` AND `parseInteger == TRUE`
short	`^[-+]?\\d*$` AND `parseShort == TRUE`
byte	`^[-+]?\\d*$` AND `parseByte == TRUE`
dateTime	`^(\\d{4})-(\\d{2})-(\\d{2})[T]? (\\d{2}):(\\d{2}):(\\d{2})[Z]?$`
date	`^(\\d{4})-(\\d{2})-(\\d{2})$`
boolean	`true|false`
anyURI	`\\b(https?|ftp|file):// [-a-zA-Z0-9+&@#/%?=~_|!:,.;]* [-a-zA-Z0-9+&@#/%=~_|]`
email	`^[_A-Za-z0-9-\\+]+(\\. [_A-Za-z0-9-]+)* @[A-Za-z0-9-]+ (\\.[A-Za-z0-9]+)* (\\.[A-Za-z]{2,})$`

elements for method requests, since requests usually have a limited number of input parameters (if any at all), and `representation` elements that refer to the XML schema `simpleTypes` for method responses, because responses may be as long as a JSON or XML file.

Naturally, it is impossible to determine that a parameter is an enumeration if a single URL is examined. This is one of the reasons why *WSDarwin* supports the batch processing of multiple URL requests. The tool keeps a record of all the values for all the parameters along with the frequency they are encountered in the given set of URLs. To determine the frequency of a value, exact matching is used rather than a fuzzy textual or numeric similarity. This is because partial matching may be coincidental, while exact matching clearly indicates that the exact same value was encountered in more than one different URLs. If, for a parameter, a single value occurs in more than one URLs, then the parameter is highlighted to inform the user that it may potentially be an enumeration. The user can then choose to confirm this inference, in which case *WSDarwin* automatically refactors the parameter into a simpleType with enumerations.

B. Resources with Variable ID

The ability of *WSDarwin* to recognize resources with variable identifiers and appropriately specify them in the WADL enables the construction of concise and compact interface specifications. The alternative would be to record each instance of a resource class (e.g. each different account identifier under the accounts resource) as a different resource, which could result in a long interface depending on the number of instances.

The identification of classes of resources occurs during the batch processing of the URL requests. The various resource paths comprising the URL are compared and counted and the tool looks for systematic differences in the same part of the URL. For every such path, its prefixes and suffixes in every URL are also checked. If the path component that differs between URLs is found to have a common prefix in all the examined URLs and a common suffix in all the URLs that are at least as long (in terms of number of path components) as the longest examined URL, then this path component is considered to correspond to a resource class. The reason for the last condition is because it is expected for the examined URLs to have different lengths (covering different parts of the API) and, thus, it is unrealistic to expect that a path will have a common suffix in all the examined URLs. Furthermore, a path is not examined with respect to its variable identifier, if it occupies the last position in the longest examined URL, since this might just correspond to different resources instead of a class of resources. Eventually, different paths that are surrounded by the same resources are clear indications of resources with variable identifiers.

Another method to confirm that a resource has a variable id, especially for resources whose path lies at the end of a URL, is by checking the responses produced by the requests. If two or more requests accessing different resources produce exactly the same responses in terms of structure, we can confirm that these requests correspond in fact to the same class of resources, which should be treated as having variable identifiers. However, the validity of this method depends on the quality and the coverage of the requests. If the requests are constructed in a way, so that they use different parameters to access the resources, then the responses might be different or they may be incomplete and the method would fail. Therefore, this method should be used as a complement to others and it should always be subjected to the user's judgement.

Once a resource with variable identifier is found, it is specified accordingly in the WADL file. A `resource` is added, whose `path` attribute is the term *resource* along with the position of the path in the URL surrounded by curly brackets, e.g. `{resource3}`. Moreover, a `param` is added in the resource with the same id as the path and its `style` is set to `template` (instead of `query`, which is the norm for input parameters). This parameter is used to indicate to the middleware, which will generate the client proxy based on the WADL interface, that a concrete identifier needs to be specified before trying to access the particular resource of the service.

C. Implementation Status

We have implemented the WADL generation on the *WSDarwin* platform. The tool is part of the *WSDarwin* Eclipse plug-in[2] for the support of the development, evolution and maintenance of service client application. As part of the Eclipse

[2]http://hypatia.cs.ualberta.ca/~fokaefs/index.php?option=com_content&view=article&id=51&Itemid=68

platform, the tool is fully integrated with the client's development environment making the provided support seamless to the general development process of the client application. The tool is also offered as part of a web application[3] specifically targeted to support REST applications. The application is not integrated to a development environment, but it can provide support for the generation and comparison of lightweight artifacts like the WADL interface and the client proxies. Both implementations of the tool are still in their beta version and we are looking towards perfecting and extending them.

IV. The Tumblr Case Study

Tumblr[4] is a microblogging platform and social-networking website, which allows its users to upload multimedia content, including text, photos, video and audio, to a short-form blog. The REST API [8] of the website exposes to client software all the data of its blogs assuming that the client application has authorization and has been granted an application key. There are two versions of the API still available; version 1[5] and version 2[6].

For our study, we prepared 7 and 16 requests for the two versions, focusing more on accessing the blog posts, since this resource seems to have the greatest correspondence between the two versions. Using these requests, we were able to produce a WADL specification for each version using *WSDarwin*[7]. We then proceed to discuss how the API changed from version 1 to version 2 based on the differences in the WADL files and what impact these changes may have on client applications.

A. WADL interface and client proxy generation

Figure 2 shows part of the WADL files (focusing on the service part and omitting the schema details) produced for the two versions of the Tumblr API by *WSDarwin*. We ran the interface generator 10 times for each version and the average execution time was approximately 9 ms (st. dev. 1 ms) for version 1 and 16.5 ms (st. dev. 3.5 ms) for version 2. These time measurements indicate two things. First, the execution time for the generator depends on the number of the input requests and the complexity of the produced interface. This is expected since the interface generator has the time complexity of a depth-first search (DFS) algorithm. The WADL interface has the structure of a tree and the complexity of the DFS is linear to the number of its edges, or in the case of WADL relative to the branching factor (how many subresources and methods each resource has) and the depth of the interface. Every time a request is processed, a tree is created and it is compared and merged to the one created in the previous step. Each node of the two trees is compared and merged once (a node is not revisited once it has been merged). Therefore, the total time for the generator equals the execution of a DFS for the interface tree times the number of the input requests. In practice, this time is negligible even for a medium-sized set of input requests which makes the generator a practical tool that can be invoked on demand and frequently without significant effort or overhead.

In version 1, as we can see from Figure 2(a), the API accesses each blog as a separate service and as a result *WSDarwin* adds two different service endpoints (`resources`) for the two blogs specified in the input requests. Unlike single resource elements that can have parameterized identifiers, service endpoints cannot be merged under a variable identifier according to the WADL schema. This can be characterized as a poor design choice for the Tumblr API, since it results in a number of WADL files (one for each blog) or in one complex one (depending on how many blogs we want to access), which is rigid and hard to maintain. This will make the use of a WADL-generated interface more problematic than helpful and, returning to our motivating arguments, it is likely to hinder the maintenance of a REST application.

Another observation that we can draw from the figure concerns the input parameters for the two blogs and the fact that they are different. This shows the sensitivity of *WSDarwin* to the input requests. The user should carefully specify these requests in order to extract as much information as possible about the API. One simple heuristic is to specify all the possible input parameters for all the requests or provide as many requests as possible with many parameter combinations.

In version 2, as shown in Figure 2(b), the blog's name is no longer the the service endpoint but a resource. As such, the various blog names can be merged under a single resource with a variable identifier. This was correctly identified by *WSDarwin* and the corresponding resource was given the variable id {`resource2`}, since it's the third path component (indexing starts from 0) in the input requests. A user can manually change the id to something more meaningful.

Another observation about version 2 is the variety and the organization of the resources. Blog posts can be accessed as a collection or by a particular media type. All resources for each individual type can be accessed with the same (or similar) set of input parameters, again depending on the quality and completeness of the input requests. According to the API documentation, the various resources return a common set of output parameters and some additional ones that are specific to the particular media type. However, in the JSON implementation of the API responses, the parameters, whether common or different, are organized under the same complex types regardless of the accessed resource. *WSDarwin* uses the "exact matching" heuristic, also used to compare different versions of the same interface [21], and as a result, it recognizes the complex types with the same name as different "views" of the same type and merges them in a single type with all the elements, common and different. This is the reason why *WSDarwin* reports the same element for the response of all methods. The "exact matching" heuristic is reinforced in this

[3] http://ssrg17.cs.ualberta.ca/wsdarwin/

[4] Description partially from Wikipedia: http://en.wikipedia.org/wiki/Tumblr

[5] https://www.tumblr.com/docs/en/api/v1

[6] https://www.tumblr.com/docs/en/api/v2

[7] All data used and produced in this study including input request URLs, generated WADL interfaces and generated client proxies can be found at: http://hypatia.cs.ualberta.ca/~fokaefs/index.php?option=com_content&view=article&id=60&Itemid=69

(a) WADL file for Tumblr API V1.

(b) WADL file for Tumblr API V2.

Fig. 2. WADL specifications produced for the Tumblr API by *WSDarwin*.

case, since the responses are examined within a single version of the same API and it is expected for elements with the same name to refer to one and the same entity.

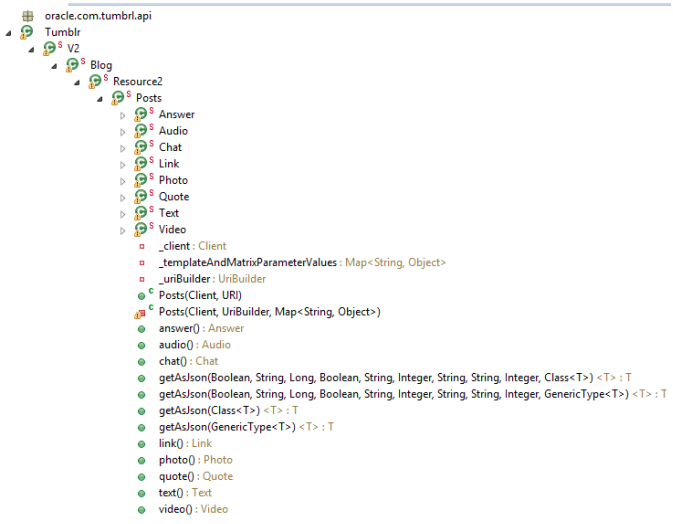


Fig. 3. The autogenerated client proxy for version 2 of the Tumblr API.

To confirm the validity of the generated WADL interfaces, we were able to generate fully compilable and executable client proxies in Java using the Glassfish WADL2Java tool [18]. The structure of the generated proxy for version 2 of the Tumblr API is shown in Figure 3. One can see the nested resources of the REST service and the `getAsJson` methods to access the `posts` resource. In order to execute the client, we first had to bind it with Jersey [22], the Oracle toolkit to develop REST services in Java as part of the Glassfish project. We were able to successfully test the clients using the input parameters.

B. The Evolution of the Tumblr API

The most significant change between versions 1 and 2 was to remedy the service-endpoint issue. In version 2, the API defines a common endpoint for all resources (and blogs) and the blog name becomes a separate parameterized resource. This is a more rational design choice and it results in a more concise service interface. Nevertheless, the change will break any clients currently using version 1, since the service endpoint changed. This change is a good example of an evolution decision with a tradeoff. On one hand, the change is incompatible with current clients, but on the other hand, it is necessary to improve the design of the service and facilitate the development and maintenance of client applications.

Another change from version 1 to version 2 has to do with how posts of different media types are accessed. In version 1, there was a “type” parameter defined in the request. In version 2, each media type is represented by a different resource within the posts resource. This may reflect a change in how the data is stored by Tumblr; in version 1, all the posts could have been stored in one resource (e.g. a database table) for each blog and then filtered by their type, while, in version 2, each type is stored in a different table for all blogs and the posts resource is a union of all the different media types. A peculiarity around this change is that the “type” parameter is retained in the requests of version 2. Although the parameter is ignored when used on a specific resource and it returns the same results when used on the posts resource, the decision to retain it in the new version was probably for consistency purposes. If a client is able to deal with the change in the service endpoint,

the old requests are not affected by the introduction of specific resources for the media types, which can still be indirectly accessed by specifying the "type" parameter.

A third interesting change that occurred in version 2 is the renaming of some input parameters. More specifically, the "start" parameter, indicating the index of the first post from which the service will start returning, was renamed to "offset", "num", which indicates how many posts the service will return, was renamed to "limit", and "tagged", which limits the response to posts with the specified tag, was renamed to "tag". Since we have to deal with simple types (strings and numbers), it is impossible to map the parameters between the two versions simply by comparing their names and structures. In such a case, we have to confirm that the renamed elements correspond to the same parameters by comparing their values in the two versions, as in the method we proposed in our previous work to map different services [23]. If two parameters of a request have the same value in both versions, then they are mapped and labelled as renamed.

Some request parameters that were added in version 2, like "reblog_info" and "notes_info", are not required and, therefore, this change will not affect existing requests. However, the "api_key" parameter that was also added is considered required and therefore old requests have to be changed in this case. This change, although breaking for current clients, was necessary to increase the API's privacy and security properties.

V. Related Work

Our work is mainly related to the development and evolution of REST applications. With the rise of REST services as dominant software components, the interest for their evolution and their software ecosystems has also peaked and this has spawned a number of research works around these topics. More specifically, in this work, we discuss a number of empirical studies around the evolution of REST and web APIs.

Wang et al. [24] perform an empirical study on how REST APIs evolve. They take the social approach and argue about the changes on a variety of REST APIs based on the discussions these changes raised among client developers in StackOverflow. The authors recognize similar changes to those we report in this paper, mainly changes in the requests and the responses of the APIs. Additionally, they report changes in the authentication method of the API, which can be considered a special change in the request of the API and changes in the rate limit, i.e. how often a client can access the API, which is usually something that is specified in the Service Level Agreement (SLA) and not in the service interface. According to the findings of the study, adding new methods raised the most questions in StackOverflow, although without a clear justification by the authors, while deleting existing methods produced the longest discussions, since this is a breaking change.

Li et al. [25] present another empirical study on the evolution of web APIs how it affects the clients, but they focus more on the technical aspects of the problems. Their data and arguments are derived by examining the native client development toolkits on various programming languages provided by the service vendors. This way they can actually argue on the impact of the changes on client applications and about which ones of them can be addressed automatically by client developers. Once again their findings are in accordance to our claims and our findings in our previous work on the evolution of SOAP services [19] both in terms of the identified changes as well as about their impact.

Finally, Espinha et al. [26] present a very complete study on the evolution of web APIs both from a social and a technical perspective. They conduct interviews with developers of large and popular REST APIs and separate interviews with developers of client applications of these APIs. Among other things, these interviews revealed certain evolution policies from the providers, like Twitter's blackout tests and Google's extensive grace period for client migration to a new version of the API, policies which were generally received with appreciation by the client developers. Then, the authors examined two open source web APIs and their clients, which are also open source. The access to the source code of both the services and the clients gave the opportunity to the authors to examined the dependencies between the two software systems, when the service evolves. This study confirmed two of our most important claims; first, that there are great variations and inconsistencies on how REST applications are developed and evolve and, second, that these evolution strategies actually create strong dependencies between providers and clients.

Another set of relevant works concerns our contribution of an automatic tool to generate WADL service interfaces for REST APIs. RestDescribe[8] is a web application developed by Thomas Steiner [20]. The tool works very similarly to WSDarwin; it receives one or more URL requests, which are parsed to generate the WADL interface. The interface is presented in an editor, so that the user can manually change the WADL by adding, removing or changing elements. Finally, the tool offers the generation of a client proxy based on the generated proxy on a number of programming languages. Despite their similarities, RestDescribe has several shortcomings compared to WSDarwin. First, it does not exercise the service with the provided requests and as a result no response element is produced, and by extend no schema is inferred for the service, although there is functionality to infer a schema if the user manually provides the response elements. Second, when batch analysing requests, the tool doesn't merge common elements but rather appends the analysed elements in the same WADL. As a result, the tool is also incapable to recognize resources with variable identifiers.

Another tool that has the capability of producing WADL interfaces is soapUI [27]. Unlike RestDescribe, soapUI does exercise the REST service given a URL request and as a result it can infer the XML schema of the service. However, unlike WSDarwin, soapUI can only process one request at a time and it has no batch processing option. Furthermore, the capabilities of the tool stop at the schema inference and it cannot produce

[8]http://tomayac.com/rest-describe/latest/RestDescribe.html

a complete WADL from the URL request. Finally, soapUI cannot recognize certain types that WSDarwin can, including `anyURI` and `email`.

VI. Conclusions and Future Work

In this work, we present an overview of the REST architectural style for service-oriented systems and its principles and discuss how these principles are practically implemented in real-world REST services and applications. We argue that, in practice, these principles are only followed in a rather *ad hoc* manner and the development of REST applications could benefit from systematic and standardized methods and tools.

Of particular interest is the ability to efficiently and effectively support the development and maintenance of REST applications. Towards this end, we also present a tool for automatically generating WADL specifications for REST services, as part of the *WSDarwin* platform. The tool analyses REST requests and the corresponding responses, to infer a specification of the service interface. The client can then use this specification to perform a number of tasks, including to generate client proxies for a variety of programming languages, and to compare different versions of the service to understand its evolution.

To confirm that our argument about the deviation between theory and implementation of REST principles stands on valid grounds, we discuss the case of the Tumblr API and study its two versions. Our study indicates that Tumblr is a typical case of a REST API, whose original version was rather poorly designed and that it was greatly improved in the next version when it complied better with the REST principles. Tumblr also gave us the opportunity to demonstrate the *WSDarwin* WADL generator and show how such a tool can help us better study the history of a REST API.

Although we have built prototypes of the WADL generator, the tool is still in its infancy. We plan to further develop it, test it with more APIs and use it to complement the rest of the *WSDarwin* platform and other third-party tools in order to offer a more complete solution for the support of REST application development and maintenance. Next, we plan to use the more complete and robust *WSDarwin* toolkit to conduct an extensive empirical study on how REST services are implemented and evolve and how much they deviate from theoretical principals, if at all. This study will be along the lines of our previous work on SOAP service [19].

Acknowledgment

The authors would like to acknowledge the generous support of NSERC, iCORE, and IBM.

References

[1] Monsoon Stone Edge User Forum, "Amazon soap being discontinued," http://www.stoneedge.net/forum/pop_printer_friendly.asp?TOPIC_ID=12687, June 2011.

[2] E. Tholomé, "A well earned retirement for the soap search api," http://googlecode.blogspot.ca/2009/08/well-earned-retirement-for-soap-search.html, August 2009.

[3] T. Anderson, "Ws-* vs the rest," http://www.theregister.co.uk/2006/04/29/oreilly_amazon/, April 2006.

[4] A. Trachtenberg, "Php web services without soap," http://www.onlamp.com/pub/a/php/2003/10/30/amazon_rest.html, October 2003.

[5] M. Hadley, "Web application description language," http://www.w3.org/Submission/wadl/, August 2009.

[6] A. R. Roberto Chinnici, Jean-Jacques Moreau and S. Weerawarana, "Web services description language (wsdl) version 2.0 part 1: Core language," http://www.w3.org/TR/wsdl20/, June 2007.

[7] Twitter, "REST APIs," https://dev.twitter.com/rest/public, 2014.

[8] "Tumblr API," https://www.tumblr.com/docs/en/api/v2.

[9] G. Developers, "The Google Geocoding API," https://developers.google.com/maps/documentation/geocoding/, October 2014.

[10] RottenTomatoes, "API Overview," http://developer.rottentomatoes.com/docs.

[11] J. McKendrick, "Cloud Computing's Vendor Lock-In Problem: Why the Industry is Taking a Step Backward," http://www.forbes.com/sites/joemckendrick/2011/11/20~/cloud-computings-vendor-lock-in-problem-~why-the-industry-is-taking-a-step-backwards/, November 2011.

[12] M. Fokaefs and E. Stroulia, "The WSDarwin Toolkit for Service-Client Evolution," in *Proceedings of the 2014 IEEE Internation Conference on Web Services, Work In Progress (ICWS'14 WIP*. Anchorage, Alaska, USA: IEEE, 2014, pp. 716–719.

[13] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[14] C. House, "How restful is your api?" http://www.bitnative.com/2012/08/26/how-restful-is-your-api/, August 2012.

[15] P. Kaminski, M. Litoiu, and H. Müller, "A design technique for evolving web services," in *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research - CASCON '06*. New York, New York, USA: ACM Press, Oct. 2006, p. 23.

[16] M. Fokaefs, E. Stroulia, and P. R. Messinger, "Software Evolution in the Presence of Externalities: A Game-Theoretic Approach," in *Economics-Driven Software Architecture*, I. Mistrik, R. Bahsoon, R. Kazman, K. Sullivan, and Y. Zhang, Eds. Elsevier, 2013.

[17] A. CXF, "wadl2java command line tool," http://cxf.apache.org/docs/jaxrs-services-description.html#JAXRSServicesDescription-wadl2javacommandlinetool.

[18] Glassfish, "wadl2java Tool Documentation," https://wadl.java.net/wadl2java.html, 2013.

[19] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau, "An Empirical Study on Web Service Evolution," in *Proceedings of the 2011 IEEE International Conference on Web Services*, ser. ICWS '11, Washington, DC, USA, 2011, pp. 49–56.

[20] T. Steiner, "Automatic multi language program library generation for rest apis," Ph.D. dissertation, 2007.

[21] M. Fokaefs and E. Stroulia, "Wsdarwin: Studying the evolution of web service systems," in *Advanced Web Services*. Springer, 2014, pp. 199–223.

[22] Jersey, "Restful web services in java." https://jersey.java.net/, September 2014.

[23] B. Bazelli, M. Fokaefs, and E. Stroulia, "Mapping the responses of restful services based on their values," in *Web Systems Evolution (WSE), 2013 15th IEEE International Symposium on*. IEEE, 2013, pp. 15–24.

[24] S. Wang, I. Keivanloo, and Y. Zou, "How do developers react to restful api evolution?" in *Service-Oriented Computing*. Springer, 2014, pp. 245–259.

[25] J. Li, Y. Xiong, X. Liu, and L. Zhang, "How does web service api evolution affect clients?" in *Web Services (ICWS), 2013 IEEE 20th International Conference on*. IEEE, 2013, pp. 300–307.

[26] T. Espinha, A. Zaidman, and H.-G. Gross, "Web api growing pains: Loosely coupled yet strongly tied," *Journal of Systems and Software*, 2014.

[27] "Soapui," http://www.soapui.org/.

3.2 Mapping the responses of RESTful services based on their values

Bazelli, B., Fokaefs, M., Stroulia, E., 2013. Mapping the responses of restful services based on their values. In: IEEE International Symposium on Web Systems Evolution (WSE 2013). IEEE, pp. 15-24.

Mapping the Responses of RESTful Services Based on their Values

Blerina Bazelli, Marios Fokaefs, Eleni Stroulia
Computing Science
University of Alberta
Edmonton, Canada
{bazelli, fokaefs, stroulia}@ualberta.ca

***Abstract*—The distributed nature of service-oriented architectures imposes some very interesting challenges to the participants of a service system, i.e., the provider and the client. For example, the service may change in a way that no longer satisfies the client's needs, either due to its reduced offered functionality or quality, due to its reduced availability or due to its increased price. In this case, the client may seek to replace the consumed service with another from a competitive provider. The client will also have the challenging task of mapping the elements of the old service to those of the new service, in order to apply the appropriate changes to the client application. In this work, we propose a novel approach to perform this mapping based on the data exchanged by the service and the application (i.e., the values of the input and the output parameters of the service). This technique allows us to avoid any potential ambiguities in the vocabulary or the structure of service interfaces between different vendors. Eventually, we evaluate the performance of our mapping technique on different services from two domains, namely movie and geolocation services.**

Keywords—service-oriented architectures; RESTful web services; service interface mapping

I. INTRODUCTION

Service-oriented architecture rose to become the prevalent paradigm for developing modular software. In the SOA style, systems are constructed as compositions of services, which, at run time, exchange data through their web-accessible XML-defined interfaces. Changes in the availability of the service, i.e., server downtime or changes in the interface or the price of the service, may cause disruptions in the proper function of a client application and may motivate the client to switch to a comparable service. To that end, three steps are necessary: a) *service discovery*, b) *service selection* and c) *service mapping*. In service discovery, the client aims to find a set of similar and eventually substitutable services from a repository that corresponds to a specific domain. With service selection, the client attempts to find the service that best satisfies his functional and quality requirements from this set of substitutable services. Finally, service mapping refers to the task of finding the correspondence between the elements of a *source service*, which the client already consumes, and the elements of a *target service*, which is the result of the selection process, in order to eventually adapt the client application from consuming the source service to now consume the target service.

In our work, we focus on the last step. The problems of service mapping and matching (i.e., the process of identifying equivalence between two services) have been extensively studied by the research community. Structure-matching techniques compare service elements according to their method signatures (i.e., number of parameters, types of parameters etc.). Across vendors, similar service elements may use different types to refer to the same concept (e.g., an integer can be expressed either as a number or as a string). Lexical-matching techniques rely on the etymology or the taxonomic and patronymic relationships of the elements' identifiers. These techniques may fail to appropriately map service elements, if the different vendors use a completely different vocabulary to refer to similar concepts. Semantic-matching techniques require semantic-web metadata for the specification of the service interfaces, which are not always available.

In this work, we propose a novel approach for service mapping that relies on the values of the data that services consume and produce at run time. This approach is particularly useful for mapping resource-based RESTful (Representational State Transfer) services [6]. These services have recently gained popularity as they rely on a stateless and simple client-server communication protocol where the HTTP protocol is used. Client applications can issue simple HTTP requests to a RESTful service in order to send, retrieve, modify or delete data.

Clients of these services acquire and manipulate data from the services, through CRUD operations. Therefore, in order to assess the degree to which two alternative services are substitutable from the perspective of a client, one needs to map the data received from the two services in question. The proposed method compares the instance values of two services and reports to the client-developers a ranked mapping so that they can a) identify the target elements that correspond to the source elements, b) know the degree of similarity between the elements and c) what the necessary transformations are, so that the instance values of the target service are translated in the values of the source service. We have evaluated our method and the accuracy of the ranked mapping in terms of (Table IV) age precision by applying it on different services from two domains, namely geolocation services and movie database services.

The rest of this paper is organized as follows. In Section II, we give an overview of the related research. In Section III, we

978-1-4799-1608-5/13/$31.00 ©2013 IEEE

present our methodology and in Section IV the evaluation of our method. Finally, Section V concludes this work with a discussion about our future plans.

II. Related Work

Significant research has been done on the topic of matching web services. Most of these works focus on the semantics [4, 8, 19] and the structure of service interfaces. Similar work on schema matching has been performed in the database field in the context of migrating one database structure to another.

A. Service Interoperability

Aversano *et al*. [1, 2] have focused on the evolution of Service-Oriented Systems. They proposed an approach, which identifies relationships among services. These relationships rely on 1) the similarity among services (when a particular service can replace others, then they are considered to be similar), 2) the inheritance factor (some services perform more abstract functionalities than others) and 3) similar characteristics (services that may share similar characteristics but cannot replace one another). In order to identify the above relationships, they define the Formal Concept Analysis tool. By using it, a lattice of semantically related words that describes the relationships among services is built. Moreover, multiple lattices not only show the relationships but also the evolution of the services being involved. However, the use of semantics may not be reliable since services from different vendors tend to use different words to describe similar functionality and also they provide a mapping on a higher level and not on a per-element basis like we do.

Wang and Stroulia [8], in their work focused on the topic of service discovery process. The methods that may lead to relevant web services categorized by their functionalities are: 1) signature matching and 2) specification matching. The former implies that, based on the WordNet[1] tool (a dictionary that contains semantically related words) applied on the Web Services Description Language (WSDL) descriptions, a group composed of relevant web services is being found. A further refinement is performed on the relevant web services found previously by applying a structure-matching algorithm which searches for similar input and output messages (by comparing the data types of the input and output parameters). A score is assigned for each pairwise comparison that indicates the degree of similarity. Finally, the pairs of services having the highest scores are considered similar.

Nonetheless, although some methods may have the same signature types, they may have different functionalities. In such cases, the semantics of identifiers are taken into account. More specifically, based on WordNet when two identifiers are identical or synonyms then a high score is being given. However, this technique is not sufficient enough as not only the identifiers must be words having a meaning but also it fails to map components that use identifiers in different languages (multiple dictionaries support).

Khorasgani *et al.* [7] aim to match RESTful web services based on the semantic similarities among web services by relying on the semantics of the service described on the WADL[2] (Web Application Description Language) files. They propose a graph-theoretic matching technique called Semantic Flow Matching (SFM). First it creates a network of WADL elements by associating them based on their structure (methods are connected to the resources where they belong to and to the parameters they include) and semantically (elements with synonym terms or identifiers having the same root are connected). The WordNet tool and the Porter stemming algorithm are used to find similar terms. However, the results are not promising even when combining these two techniques together as although some words may refer to the same concept, they may not be synonyms according to WordNet nor sharing the same root according to the Porter stemming algorithm (e.g., the words 'bookmark' and 'tag' are not synonyms according to WordNet despite the fact that they refer to the same concept).

Ponnekanti and Fox [9, 10] focus on the topic of substitution among services provided by different vendors. If a service could substitute another service, then their vendors could compete in terms of price, quality of service and availability. They propose a solution based on four incompatibilities: structural, value (e.g., unexpected forms that require input values), encoding and semantic. The structural and value incompatibilities derive generally from additions/deletions of methods and fields from source and target services. In order to identify these incompatibilities, they convert the WSDL descriptions and the XML schema types into JAVA classes and then generate usage tuples that indicate all the calls performed during while the application is running. In our work, we do not define services as incompatible based on their methods or fields used by other applications but based on their response values. Therefore, we consider two services substitutable when for a given input, their responses' values match to some extent.

While all the above work on the area of service matching focused mostly on the semantics and the structure of the web services, the results depending on these two factors are not satisfactory as there may be no commonalities among the names given to elements that constitute a web service among the vendors. Nonetheless, in our work we focus directly on the input and output values as they do not depend on the vendors.

Gokhale et al. [13] attempted to map two APIs in method level through their tool, Rosetta. First, they collect manually a number of applications having the same functionality (e.g., different implementations of the TicTacToe game). Then, a user performs manually the same functionality on both applications. On the background, the tool traces all the method calls made on both applications in order to be able to map the appropriate methods while they are executed. As a result, there is a collection of API call pairs. In addition, they take into account the number of method calls per trace. For example, three calls to method A (of application 1) and B (of application 2), indicate that these methods are more likely to be mapped. This approach differs from ours in the sense that we map the output elements of a RESTful API request.

[1] http://wordnet.princeton.edu/

[2] http://www.w3.org/Submission/wadl/

B. Schema Migration

The problem of schema matching is also applied in many other fields besides web-services such as in the filed of Databases [14, 15, 18] and Graph Theory among others. Schema matching refers to the process of matching two elements that are somehow semantically related and able to substitute one another. Shvaiko and Euzenat [3] proposed a classification of techniques applied that lead to schema matching. They rely on techniques related to ontology matching such as structural and semantic. Drumm et al. [4] proposed a semi-automatic approach to migrate data from one application's database to another. This process requires two important steps: 1) schema matching and 2) mapping discovery. Since they focus on the interoperability of legacy systems the involvement of users is unavoidable. Thus, the first step of their approach is to ask the users of the source system to provide information about the schemas. It is also important to store a sample of missing instances from the target system to the source system. By doing this, they ensure that there is definitely a mapping among elements between the two systems. Finally, several matchers are applied to pairs of source and target elements in order to map them based on their instance values (equality matcher, split-concat matcher etc.).

Rahm et al. [12] have classified the most widely used techniques that lead to schema matching. They define two main categories of automatic schema matching: 1) schema-level matching and 2) instance-level matching. The former relies on the information included in the schema. These pieces of information are being approached based on linguistic characteristics (e.g., synonyms, hypernyms) or based on constraints such as relationship types, data types etc. The instance level approach is related to the values of the schema elements and is usually used to refine the results given by the previously mentioned techniques. The techniques that prevail in instance level matching depend on information retrieval approaches for text elements and constraints for structured defined data (e.g., numerical and string elements). Finally, hybrid matchers, combine multiple techniques in order to achieve satisfactory matching results. There are two ways to perform a hybrid matching, either apply all the techniques at the same time or apply them one after another one has finished.

Islam et al. [14] have focused on the mapping in element level of large corpus text in databases by taking into consideration the elements' identifiers. They proposed a method of schema matching that uses two methods, 1) the word similarity method and 2) the word segmentation model. The first one is related to the probability of two words to co-occur within a text corpus whereas the second refers to the distinguishing of concatenated words into multiple meaningful ones. These approaches cannot be applied in our work as we map output data based on their values. The values are usually numerical or strings. Therefore there are not concatenated words where the word segmentation model could be applied.

From the database schema-matching field, the work of Miller et al. [15] is the closest to our work as they consider the values as the most important factor to map two elements originated from two different databases. They focus on the ability to transform a query created to get data from a source database to a query to get data from a target database. The tool significantly interacts with users as they need to give as input all the value correspondences (the functions that define a relationship between multiple elements) from the source to the target schema [20]. In our work, we have to deal with unstructured data values in the sense that we cannot assume any relations between them. As a result, we cannot use the same technique used by Miller et al. and for this reason we have defined our own mapping rules based only on the type of the values.

Few of the techniques described above could also be applied in the case of RESTful web service mapping (use of semantics, word segmentation). The reason is that these techniques require extensive involvement and technical/ domain knowledge by the user. Furthermore, we have already discussed the issues of using semantics in order to map web service elements.

III. METHODOLOGY

Our methodology requires as input a set of requests; more specifically, at least one request per operation for each RESTful service involved. We use these requests to invoke the services and acquire the responses. We then proceed to map the elements of the responses according to a set of pre-defined rules based on the elements' data types, i.e., different rules are applied on numerical and string elements.

We assume that the client-application developer has already identified a set of *candidate* RESTful web services from a desired domain that can potentially substitute the *source* service, i.e., the service that the client application already invokes. Based on the requests that the client applications already issues to the source service, we further assume that the client developer can formulate requests for candidate web services, which implies that the developer has to be able to map the input parameters of the source-service requests to input parameters for the requests to the candidate target services.

The *first step* of the process then is to invoke the requests for the source and target web services in order to get the response files (typically in JSON or XML format). The responses are parsed and translated in a lightweight java representation specified by the WSDarwin tool [17], in order to facilitate the comparison process.

In the *second step*, our method proceeds to map the values returned by the invoked service operations. A response element may be either a primitive type or a complex type. Primitive types are considered the following data types: Integer, Double, Long, Float, String, Boolean, Date, and URL. We observed that the URLs are often vendor specific, and therefore we decided to exclude them from our mapping process. A complex type consists of primitive types either by composition or by aggregation (i.e. sets of primitive types). The complex types are processed in a transitive manner and eventually only the primitive types participate in the mapping process.

We define several comparison rules that we apply to a pair of elements, according to their types. These rules rely on two metrics: similarity and inclusion. *Similarity* refers to how "close" two instance values are given their types. Similarity is

computed differently for strings and numerical elements. The *inclusion* criterion examines whether a sequence of characters in one service-response is included in a character sequence in another service response. The intuition for this criterion is that in RESTful services, differences in the schema of the underlying resource frequently manifest themselves as reorganizations of the values of the resource instances. For example, one resource may distinguish between "first" and "last names" where a second may simply have a single element for a "name"; in this case, two values included in separate JSON elements in the first response will be included in a single value of a single element in the second response.

The similarity between two numerical elements (Integer, Double, Float, Long) is computed as follows:

$$dist\ (n1, n2) = \frac{max(|n1|,|n2|) - min(|n1|,|n2|)}{max(|n1|,|n2|)} \qquad (1)$$

where $|n_1|, |n_2|$ are the absolute values of two numbers n_1, n_2

The range of the two numeric values n_1 and n_2 depends on the JSON elements that are being compared.

In order to compute the distance among sequences of characters (strings) we use the Levenshtein edit distance metric [11], which counts the minimum number of characters required to change one sequence of characters into the other. This similarity metric takes as input two sequences of characters (a, b) and is calculated as follows:

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & , \ \min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + [a_i \neq b_j] \end{cases} & , \ \text{else} \end{cases}$$

Although the Levenshtein distance metric gives satisfactory results for pairs of single-element strings, it fails to identify sequences of words separated by special characters (delimiters) as similar. Let us illustrate this point with an example. Let us suppose that we need to compare two address values, one for the Google geolocation service and one for the Bing geolocation service. The actual address we give as input is:

"8619 111 St Edmonton, AB".

Both services add their own defined delimiters between the spaces and the actual address is transformed as:

"8619+111+St+Edmonton,+AB" (in Google)
"8619%20111St%20Edmonton,%20AB" (in Bing)

According to the Levenshtein metric, the above addresses have a distance of 44.2%. However, the reason why the distance is that high is the use of delimiters. In order to avoid the noise of delimiters that may significantly affect the distance metric's results, we gathered several types of widely used delimiters for RESTful web services, such as +, ||, %, %20, space, tab, underscore etc. We import this list of special delimiters into our implementation and, if any of them are found in the compared strings, they are removed. In the above example, the addresses will be transformed into:

8619111StEdmonton,AB
8619111StEdmonton,AB

This time the Levenshtein metric returns a distance of 0%, which implies a perfect similarity. Clearly, different service providers can define their own delimiters, which may not be included in the above list. Still the above set is quite comprehensive and, even with the appearance of more rare delimiters, a low Levenshtein distance indicates that the values are very likely to be the same. The similarity (%) is given by the following formula:

$$similarity = 100(1 - dist)$$

where dist is the distance computed in (eq.1) for numerical elements or the Levenshtein distance for string elements.

In order to map the response elements, all pairwise combinations of element values are compared. Since the primitive types of the pair under comparison may differ, we have defined the following heuristic rules to cover all the possible data type combinations. We explain the heuristics in terms of examples shown in Table I.

***String vs. String*:** When the two elements to be compared are both strings, they are both transformed in lowercase, in order to make them consistent and avoid spurious mismatches, as in the case of "Movie" and "movie" for example. Then, we calculate the Levenshtein distance, which shows how different two strings are. Furthermore, we have set "Inclusion" to "TRUE" when one of the two string elements to be mapped is included into the other.

String vs. Number: In the case of a string-to-number comparison, the inclusion of a numerical value in the string is examined. More specifically, our proposed methodology starts by extracting all the number sequences from a string and storing them as a set of numbers. Next, it compares each element of this set with the target number and if they are the same, they are mapped (Table I, Row 3). This heuristic accounts for the cases where one resource associates units to the value of an element property while others do not.

***Number vs. Number*:** According to our methodology, we map numerical elements based on their maximum computed similarity. A numerical element may be an Integer, Double, Float or Long. As we can observe in the example shown in Table I Row 4, the parameters "release_year" and "year" are mapped with a similarity of 100%. In Table I Row 6, the elements "average score" and "rating" are also mapped as they have a similarity of approximately 86%, which is the maximum similarity that can be accomplished when attempting to map each of these elements against any other numerical element of the other service. Even though numbers could, in principle, be viewed as strings and the inclusion heuristic could be applied, we do not consider inclusion when comparing numbers. If two numbers refer to the same property, then they are likely to be close, barring differences due to precision variations. Consider for example the variations across geolocation service providers (Table I Row 11). These cases can be correctly mapped through the number similarity criterion above.

***Date vs. Date*:** The implementation of our methodology recognizes many typical formats of Date types, which it translates into a common format. Once an element has been recognized as a Date, it is compared only against other Date elements, and a pair is considered mapped only if the inclusion

is "TRUE" i.e., the dates refer to the same point in time (Table I Row 2).

***Set vs. Set*:** According to our technique, we compare sets that contain elements of the same data type. The similarity between two sets of strings is computed as follows:

$$similarity\ (S1, S2) = 100(1 - \frac{|S1 \cap S2|}{min(|S1|, |S2|)})$$

where S1, S2 two sets containing string data types and |S1|, |S2| the number of elements in S1 and S2 respectively.

An example of a mapped pair of sets of strings is shown in Table I Row 1. In this case, all the actors' names contained in the source set "name" are included in the set "actors". Since all of them are included, the similarity is 100%. Two Set elements are mapped when the similarity is the maximum among all the pairs where the source Set element participates in.

In cases where we have two Set elements of numerical data types, our implementation performs all the distance computations among all the combinations among the source and target elements in a similar way we compute the distance for numerical elements (1). We encountered many such cases within the responses of the geospatial services. Two sets contained the latitude and longitude values are mapped by comparing each element of the source element to each one of the elements contained into the target Set element (Table I Row 10). Each source element is mapped with a target element only if at least one value in the source set has a maximum similarity with a value from the target set.

Boolean vs. Boolean: The presence of boolean data type elements is challenging as they have a very limited range of values (true, false). Two boolean elements set to TRUE or FALSE may (or may not) refer to two semantically related elements. Therefore, we exclude comparisons of cases where two elements are of boolean type.

TABLE I. ILLUSTRATIVE EXAMPE OF THE OUTPUT OF THE MAPPING PROCESS

Row	Type	Source Element	Source Value	Target Element	Target Value	Similarity (%)
1	Set-Set	name	[Carrie Underwood, Lorraine Nicholson, Annasophia Robb, Helen Hunt, Dennis Quaid]	Actors	[Carrie Underwood, Chris Brochu, AnnaSophia Robb, Sonya Balmores, Lorraine Nicholson, Craig T. Nelson, Dennis Quaid, Helen Hunt…]	100.00
2	Date-Date	release_date	20110408	Theater	2011-04-08	100.00
3	String-Integer	runtime	USA: 106 min	Runtime	106	100.00
4	Integer-Integer	release_year	2011	Year	2011	100.00
5	String-String	title_localized	Soul Surfer	Title	Soul surfer	100.00
6	Double-Double	average_score	5.95	Rating	6.9	86.23
7	Integer-String	imdb	1596346	imdb_id	tt1596346	100.00
8	Integer-String	id	771037147	imdb_id	tt1596346	20.00
9	String-String	surname	McNamara	Type	M	12.50
10	String-String	length	null	runtime	USA: 106min	0.00
11	Double-Double	coordinates	[-113.5177994, -113.5173264, 53.5225983]	lng	-113.518802	99.98

IV. EVALUATION

Our methodology is useful under the assumption that a web-service client has already identified a family of potentially substitutable set of services. Given the need to substitute a particular service with another from this family in a client application, our technique offers client developers the ability to select the service that requires the minimum manipulation of its response data in order to be transformed into the response data of the original service used by the client application.

The method described in Section III can be implemented as a tool in two ways: (a) either as a fully automated tool where an element can be mapped only with one other element or (b) as an interactive tool which suggests several mappings between a source element and elements from the target service according to their values. In the second case, the user of such a tool can assess not only the similarity between the elements of the two services, but also the data in his disposal to call and process the output of the target service. For the purpose of our evaluation, we implemented the proposed approach as an embedded feature into the WSDarwin tool [5] and evaluated its results for both implementation scenarios.

We evaluated the effectiveness of our methodology by mapping the responses of RESTful services from two service families, in the domains of maps and movies. In the first domain, we examined (a) Google Maps[3], (b) Microsoft Bing Maps[4] and (c) CloudMade Maps[5]. In the second domain, we chose to examine (a) the Internet Movie Database (IMDb)[6], (b) Rotten Tomatoes[7], (c) Filmaster[8] and (d) TheMovieDB[9].

[3] https://developers.google.com/maps/

[4] http://www.microsoft.com/maps/developers/web.aspx

[5] http://maps.cloudmade.com/

[6] http://www.imdb.com/

[7] http://developer.rottentomatoes.com/docs

[8] http://filmaster.org/

[9] http://www.themoviedb.org/

As we have already mentioned, our methodology only compares the responses of two services, mainly because this will greatly facilitate he migration of a client from a source service to a target, but also because input mapping is rather challenging. The reason is that while we can have two responses for the two responses, we need to construct the input of the target service. This could be achieved by generating requests based on all combinations of source values with target parameters until a meaningful response is obtained. However, this may be practically impossible since many service providers typically cap the number of requests that a client may issue and sometimes an authentication step is required in order to access a vendor's API. For instance, Google allows 2,500 geolocation requests per day and up to 100,000 requests per day for business purposes. However, Google does not track the requests based on any specific API key but on IP addresses. On the other hand, Bing requests have a required query parameter referred to the API key. All the clients who want to use the web services provided by Bing should register and obtain a key. Some of the types of keys provided by Bing are: trial, basic key, key for education or non-profit purposes. The limit of the requests per day depends on the type of the key that the client had chosen when he first registered. As noted on both Google and Bing terms of use website, the numbers related to the requests are not fixed and may change at any time.

Another challenge in mapping the input of two services is that some of them may require special authentication data. For this study, we assume that the client knows whether a vendor needs their clients to authenticate/purchase an API key in order to use their services or not. Therefore, this piece of information and consequently the parameter related to authentication should be known before the invocation of the web service. The Google service does not need an authentication key and it takes only two parameters, one for the address and one for the sensor (which indicates whether the application is using a locator such as a GPS or not). On the other hand, Bing and CloudMade requests may take several parameters (not all of them are required) and among all the key parameter, which is an identification sequence of characters obtained by the client through the developers' website. Three examples of service invocations are shown below. The parameters are highlighted as bold.

Google request:

http://maps.googleapis.com/maps/api/geocode.json?**address**=8619+111+St+NW,+Edmonton,+AB&**sensor**=false

Bing request:

https://dev.virtualearth.net/REST/v1/Locations.json?**CountryRegion**=CA&**adminDistrict**=AB&**locality**=Edmonton&**addressLine**=8619%20111St.%20Edmonton,%20AB&**postalCode**=t6g2w1&**key**=APIKEY

CloudMade request:

http://geocoding.cloudmade.com/APIKEY/geocoding/v2/find.json?**query**=8619+111+St+NW,Edmonton,AB

A. Precision & Recall

We measured the performance of our methodology in terms of precision and recall. In the geolocation domain, we have six pairs of service providers (Table II). In the movies domain, four service providers lead to twelve combinations for which the precision and recall are presented in Table III. In order to compute the precision and recall, we first count the source and target elements that have been mapped correctly (manually and automatically matching) (True-Positive: TP). Then we count the parameters that have been wrongly mapped as correct ones (our implementation shows that a source element has been mapped with a target parameter, whereas we have mapped it manually with another element (False-Positive: FP)). Finally, we count the elements that we have mapped manually and our implementation reveals that the particular elements cannot be mapped with any of the target elements (False-Negative: FN). Therefore, we compute the precision and recall as:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

1) One to One Mapping

In order to manually map the responses of two requests corresponding to different service providers, one should have to look at them side by side and decide which parameters of one service provider correspond to which parameters of the other. In the geolocation domain, the parameters address (Google) and addressLine (Bing) refer to the same element and must be mapped. Moreover, the rest of the parameters for Bing do not add any new information but split the information into categories (a reason may be the fact that by splitting an address into lower level components such as country, city etc. may improve significantly the performance as an address can be found much faster; the same technique is being applied in telephone numbers).

Furthermore, by inspecting the results of each request, we can see the fields of latitude-longitude for both service providers. For the same address, the point specified on the map for each service provider may be the same or very similar (agreeing up to the 5th decimal).

Since the output may be a long response with several parameters, we compare all the "allowed" combination values. We consider a combination as "allowed" when we can apply one of the heuristics explained in our methodology section. In Figure 1 we can see a partial response of the Google and Bing services.

The manual inspection is being performed on all the combinations of the response elements. The amount of data returned by the services is an important factor as the user must be able to get back "enough data" in order to support the client application's functionality. Therefore, although we select the maximum similarity for a parameter, when the maximum value is found to occur more than once, we report all the mapped pairs, in case when a single value may be reused for multiple purposes. Therefore our implementation reports all the combinations along with their similarity scores. On the opposite side, our methodology does not report a mapping

when the distance between two values is zero and/or the inclusion criterion is false, where applicable.

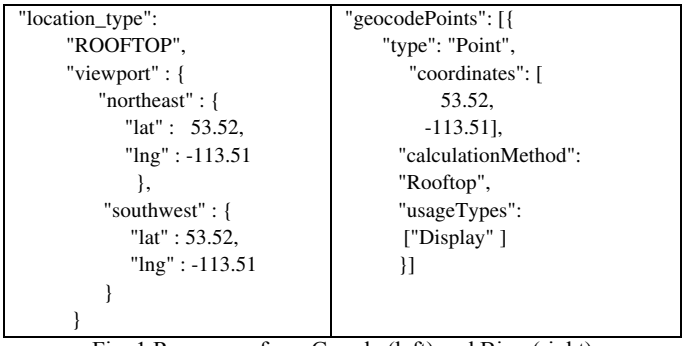


Fig. 1 Responses from Google (left) and Bing (right)

a) Geolocation Domain

The precision and recall results for the geolocation domain are presented in Table II. Overall, the average precision and recall are 67.10% and 84.52% respectively. The highest precision is observed for Google and Bing (90.91%) and the lowest for the pair of CloudMade and Google (33.33%). The high precision above is due to the fact that Google uses multiple sets to store instance values. These values are being mapped with multiple output elements of the Bing geolocation service. Since we aim to assist clients on the data construction, "one to many" mappings are allowed only when the maximum values are all the same. The low precision between CloudMade and Google is due to a single numerical data type element, which is included into the address given as input. Since there are many elements that have as value the address, this comparison is performed between all of them. Consequently, there are multiple FPs, which result into a low precision value. The recall measurements range from 66.67% to 100%. The reason behind the low recall values is that CloudMade and Google are not highly compatible. Only three elements can be manually mapped whereas two of them are correctly mapped automatically.

b) Movie Domain

For the movie domain, we have selected the "search by movie" operation, which typically takes as parameter a movie and gives as response details related to it. Both IMDb and Rotten Tomatoes support this operation; however, IMDb does not require any authentication whereas Rotten Tomatoes requires from their clients to first register in order to obtain a key, which is a required parameter in the requests. Therefore, after registering and obtaining a key, we were able to create several requests. Below is an example of such a request for IMDB and Rotten Tomatoes respectively:

http://imdbapi.org/?**title**=hunger+games&**type**=json&**limit**=10&**lang**=en-US

http://api.rottentomatoes.com/api/public/v1.0/movies.json?**apikey**=hqmxm7bm2r8bme9dzcs2&**q**=Soul%20Surfer&**page_limit**=10

The JSON responses for both IMDb and Rotten Tomatoes contain information about the year the movie was released, the country, the cast, rating etc. We first map manually the response parameters by getting the list that combines all the response elements from our implementation in order to calculate the metrics of precision and recall.

As we can see in Table III, the maximum precision and recall is 100% between IMDb - RottenTomatoes and TheMovieDB - IMDb. The main reason for low values in precision (such as in the case of RottenTomatoes and TheMovieDB or between IMDb and TheMovieDb where the precision is 33.33%) is that when we compare numbers we translate the maximum similarity value of the mapping process into a pair of mapped parameters. However, the maximum value may be very low (e.g., a similarity value of 2.59E-07 has led to a wrongly mapped pair of parameters as it was the maximum one). The results reported may be reviewed manually by the user who will disambiguate such cases and discard the wrong mappings. Our methodology achieved a precision of 100% between RottenTomatoes and IMDb. According to ProgrammableWeb[10] these two service providers are very popular and receive a large number of visitors on daily basis. When two service providers are both popular, then they tend to be more competitive i.e., when one adds a certain functionality (e.g., rating, ability of users to write comments on movies) the other one responds by adding a similar functionality, especially when it is successful and attracts more users. Therefore, these service providers tend to be more compatible, offering not only very similar operations but also similar data through their APIs. Consequently, our implementation's results can be used as a guide for checking the compatibility among web services in terms of data. The mapping of multiple source elements to multiple target elements indicates that the two services are highly compatible, as opposed to pairs of services where only few elements can be mapped.

In order to normalize the results, after the user discards and disambiguates incorrect mappings, they may calculate the numbers of parameters mapped, divided by the total number of parameters of both the source and target web services. By doing so, they have an overview of the extent to which the two web services are compatible with each other. A low value of this metric implies that although two services are presented to have similar functionalities, they are not compatible, in the sense that they do not provide the same data and therefore the migration from one service to another is likely to be too difficult or even impossible. In such cases clients should look for other web services that would probably offer the same data.

2) One to Many Mapping

Since we focus on RESTful web services, mapping of one element with many is equally important as the one to one. In this case, we select the top ten results based on the similarity metric. After removing all the duplicates (if there is any) we present these results to the users for disambiguation purposes. We calculate the mean average precision (MAP) taking into

[10] http://www.programmableweb.com/

consideration the number of correct mappings included into the top ten results, according to (2).

$$MAP = \frac{\sum_{p=1}^{P} AvePre(p)}{P}, \quad (2)$$

where p is a pair of elements, P the number of elements that can be mapped and AvePre the average precision per element

a) Geolocation Domain

Table IV shows the values of MAP for each service provider combination. As we can see, the MAP ranges from 17.85 to 79.52%. The highest values are observed between Google and Bing whereas the lowest occur between CloudMade and Google/Bing. A possible explanation is that Google and Bing are more popular compared to CloudMade, which is less compatible as many elements cannot be mapped.

b) Movie Domain

The mean average precision for the movie domain ranges from 33.33% for the pair of TheMovieDB-IMDB and 80% for the pair of Filmaster-TheMovieDB. The reason behind the low values is the presence of numerical elements. Based on the rules defined, our methodology relies only on the maximum value of the similarity metric when examining numerical values. Moreover, we do not employ any thresholds, since it is impossible to find a global threshold value for all elements and across domains.

B. Threats to validity

We chose two domains for our evaluation namely, geolocation and movie service. Both these domains are related to ground truth information. For instance, a movie has a director, a list of actors and a production country. Two services that provide movie information would provide the same information for a particular movie. We ran our experiments on three service providers for the first domain and four for the second domain. At this point, we cannot generalize our results for every possible domain as there may be a selection bias while choosing the above service providers. However, in order to mitigate these threats to validity we plan to run more experiments on different domains in the future.

In addition, it can be argued that we have experimenter bias since the evaluation was performed by the authors. We tried to avoid the bias in our results by applying the manual mapping before the automatic mapping. Finally, it should be noted that the manual mapping was performed by the developers of the implementation although we tried to be as objective as possible. This problem would have been mitigated, if we had assigned the manual mapping to people who had no knowledge about the implementation and most importantly the rules being applied to map the elements.

C. Limitations

An important problem we encountered was the null values of several parameters. Since our approach relies on the output values, we miss the cases where a parameter value is null. For example (Table I Row 10), we have manually mapped the parameter "runtime" of IMDb with the parameter "length" of Filmaster. However, our implementation is not able to give us a strong similarity score for these elements because the value of the former is "USA:106 min" whereas the value of the latter is set to "null". We noticed that the null value of the parameter "length" is not always null by invoking a new request after changing the query parameter with another movie title. This problem may easily be solved if the user gives a different request associated with another movie. However, the cases of null values are important, especially when they occur in the web services considered as candidates for substituting a service. If the client application needs this missing piece of information, then the target web service is not suitable, as although it may perform a similar functionality with the source target, it cannot replace it in terms of data.

Moreover, there are URLs and IDs specific to service providers, which cannot be mapped since their values change from one service provider to another. In Table I Row 8 we can see that the RottenTomatoes ID value is different from the ID of the IMDb. The comparison between these two elements gives a similarity of 20%. Therefore, although the elements "id" and "imdb_id" are semantically related as they refer to the same concept (i.e., a unique identifier internal to each of the two providers), our technique is not able to map them. However, we are able to map IDs that refer to the same service provider (Table I Row 7). For instance, since IMDb is considered one of the most popular movie information related website, some other web-services who offer movie information data include in their dataset the IMDb ID. In this case, the number included in the String "tt1596346" is extracted and compared with the integer value of the "imdb" element. Since they have a similarity of 100, they are successfully mapped.

Single letter instance values may cause errors in the sense that there is a high probability a single letter word to be included into another sequence of characters. An example is given in Table I Row 9 where the surname of the director of a movie is McNamara and an element's value on the target element is "M" that refers to the type of the entry, which is a movie. Although the similarity value between these two strings may not be considered high, it is the maximum among the other comparisons and the inclusion value is set to TRUE. As a result, we report this pair of strings as mapped to the user who has to disambiguate it.

We report the results to the users as shown in Table I. However, we do not report the cases shown in Rows 8, 10. We have added these cases to Table I only for the purpose of illustration special cases we run into while running our experiments.

Although the results of our methodology may be greatly improved by manual inspection from the user, especially in the case of ranked results, our experiments have shown that the total manual effort required is 7 (for the movies domain) or 6 (for the geolocation domain) times less than the effort required to completely manually map the service responses (Table V).

TABLE II. PRECISION & RECALL FOR THE GEOLOCATION DOMAIN

Source / Target	Google					Bing					CloudMade				
	TP	FP	FN	P (%)	R (%)	TP	FP	FN	P (%)	R (%)	TP	FP	FN	P (%)	R (%)
Google						10	1	1	90.91	90.91	2	3	1	40.00	66.67
Bing	13	3	1	81.25	92.86						11	2	0	84.62	100.00
CloudMade	2	4	1	33.33	66.67	9	3	1	75.00	90.00					

TABLE III. PRECISION & RECALL FOR THE MOVIE DOMAIN

Source / Target	IMDb					RottenTomatoes				
	TP	FP	FN	P (%)	R (%)	TP	FP	FN	P (%)	R (%)
IMDb						7	3	3	70.00	70.70
Rotten Tomatoes	8	0	2	100.00	80.00					
Filmaster	11	4	3	73.33	78.57	4	5	3	44.44	57.14
TheMovieDB	5	2	1	71.43	83.33	3	6	1	33.33	75.00
Source / Target	**Filmaster**					**TheMovieDB**				
	TP	FP	FN	P (%)	R (%)	TP	FP	FN	P (%)	R (%)
IMDb	9	2	2	81.82	81.82	3	6	0	33.33	100.00
Rotten Tomatoes	4	2	3	66.67	57.14	3	3	1	50.00	75.00
Filmaster						5	3	1	62.50	83.33
TheMovieDB	4	5	2	44.44	66.67					

TABLE IV. MEAN AVERAGE PRECISION

(a) Geolocation Domain

Source / Target	Google	Bing	CloudMade
	P (%)	P (%)	P (%)
Google		79.52	25.00
Bing	58.62		17.85
CloudMade	20.00	30.83	

(b) Movies Domain

Source / Target	IMDB	RottenTomatoes	Filmaster	TheMovieDB
	P (%)	P (%)	P (%)	P (%)
IMDB		57.16	67.00	67.00
RottenTomatoes	58.33		38.00	35.00
Filmaster	75.00	57.14		80.00
TheMovieDB	33.33	50.00	74.00	

TABLE V. COMPARISON OF THE MANUAL EFFORT NEEDED TO MAP TWO SERVICE RESPONSES TO THE MANUAL EFFORT NEEDED TO REFINE THE IMPLEMENTATION'S RESULTS

(a) Geolocation Domain

Manual Mapping Effort (in min)			
Source / Target	**Google**	**Bing**	**CloudMade**
Google		2.47	3.14
Bing	4.23		3.28
CloudMade	3.17	3.29	
Manual Refinement Effort (in min)			
Source / Target	**Google**	**Bing**	**CloudMade**
Google		0.51	0.37
Bing	1.23		0.47
CloudMade	0.28	0.35	

(b) Movies Domain

Manual Mapping Effort (in min)				
Source / Target	**IMDB**	**RottenTomatoes**	**Filmaster**	**TheMovieDB**
IMDB		4.48	5.46	3.49
RottenTomatoes	3.38		4.15	2.08
Filmaster	3.27	4.37		2.49
TheMovieDB	3.05	3.43	2.53	
Manual Refinement Effort (in min)				
Source / Target	**IMDB**	**RottenTomatoes**	**Filmaster**	**TheMovieDB**
IMDB		1.18	1.14	0.28
RottenTomatoes	0.43		0.44	0.33
Filmaster	0.56	0.36		0.31
TheMovieDB	0.50	0.26	0.22	

V. CONCLUSIONS AND FUTURE WORK

Within the context of this work, we aimed to map service response elements based on their instance values by following several rules we defined. These rules rely on the data types of the elements to be mapped. We implemented the methodology on the WSDarwin tool [17]. Our approach assumes that users have already mapped input elements and can create the appropriate requests to invoke source and target web services. Our contribution aims to assist clients as to what changes need to be made in the processing of the output data of the service.

Several studies have tried to match web services based on their semantics (by utilizing dictionaries such as the WordNet tool and different kind of algorithms such as the Porter stemming algorithm) and the structure of the elements included in the WADL files. However, while they focus on mapping services, we focus on mapping output data elements, which is equally important. When a client decides to migrate from one service to another, they need to know that the target service provides the data they need to either view or process.

We chose two different domains (geolocation and movies) in order to evaluate our methodology. For each one of these domains, three and four service providers had been chosen respectively. The results are promising as our tool achieved high values of precision and recall among service providers.

In the future, we will be focusing on integrating our work with other techniques that have already been implemented such as mapping by IDs (semantics) and mapping by structure (e.g., parameter data types). Moreover, we would like to run experiments by changing the order of these techniques and discover which one gives better results. For example, we would like to see if results are improved when performing the mapping process firstly by value, then by structure and finally by ID. Furthermore, we plan to apply the above methodology into more complicated web services in terms of operations (e.g., services that include several methods) and on services that are not "ID focused".

REFERENCES

[1] Aversano, L., Bruno, M., Di Penta, M., Falanga, A., & Scognamiglio, R. (2005, September). Visualizing the evolution of web services using formal concept analysis. In Principles of Software Evolution, Eighth International Workshop on (pp. 57-60). IEEE.

[2] Aversano, L., Bruno, M., Canfora, G., Di Penta, M., &Distante, D. (2006). Using concept lattices to support service selection. International Journal of Web Service Research (IJWSR), 3(4), 32-51

[3] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," in Journal on Data Semantics IV, ser. Lecture Notes in Computer Science, 2005, ch. 5, pp. 146–171.

[4] Drumm, C., Schmitt, M., Do, H. H., & Rahm, E. (2007 November). Quickmig: automatic schema matching for data migration projects. In Proceedings of the sixteenth ACM conference on Conference on information and knowledge management (pp. 107-116).

[5] Fokaefs, M., & Stroulia, E. (2012, November). WSDarwin:automatic web service client adaptation. In Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research(pp. 176-191). IBM Corp.

[6] Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures* (Doctoral dissertation, University of California).

[7] Khorasgani, R. R., Stroulia, E., & Zaiane, O. R. (2011, September). Web service matching for RESTful web services. In Web Systems Evolution (WSE), 2011 13th IEEE International Symposium on (pp. 115-124). IEEE.

[8] Wang, Y., Stroulia, E.: Semantic structure matching for assessing Web-service similarity. In: Proceedings of First International Conference on Service Oriented Computing (ICSOC03), pp. 194–207. Springer, Berlin, 2003

[9] Ponnekanti, Shankar R., and Armando Fox. "Interoperability among independently evolving web services." Middleware 2004. Springer Berlin Heidelberg, 2004. 331-351.

[10] Ponnekanti, S. R., & Fox, A. (2003, March). Application-service interoperation without standardized service interfaces. In Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on (pp. 30-37). IEEE.

[11] Heeringa, W. J. (2004). Measuring dialect pronunciation differences using Levenshtein distance (Doctoral dissertation, University Library Groningen][Host]).

[12] Rahm, E., & Bernstein, P. A. (2001). On matching schemas automatically.*VLDB Journal*, *10*(4), 334-350.

[13] Gokhale A., Ganapathy V., Padmanaban Y. Inferring Likely Mappings between APIs. In the proceedings of the 35th International Conference on Software Engineering, 2013.

[14] Islam, A., Inkpen, D., & Kiringa, I. Database Schema Matching using Corpus-based Semantic Similarity and Word Segmentation.

[15] Miller, R. J., Haas, L. M., & Hernández, M. A. (2000, September). Schema mapping as query discovery. In *Proceedings of the 26th international conference on very large data bases* (pp. 77-88).

[16] Pathak, J., Koul, N., Caragea, D., & Honavar, V. G. (2005, November). A framework for semantic web services discovery. In *Proceedings of the 7th annual ACM international workshop on Web information and data management*(pp. 45-50). ACM.

[17] M. Fokaefs and E. Stroulia, "WSDARWIN: Studying the Evolutionof Web Service Systems." in Advanced Web Services,B. Benatallah, M. S. Hacid, A. Leger, C. Rey, , and F. Toumani,Eds. Springer Berlin / Heidelberg, 2013, ch. 9.

[18] Doan, A., Noy, N. F., & Halevy, A. Y. (2004). Introduction to the special issue on semantic integration. *ACM Sigmod Record*, *33*(4), 11-13.

[19] Li, N., & Cai, H. (2009, October). Functionality semantic indexing and matching method for RESTful Web Services based on resource state descriptions. In*Computer Science and Engineering, 2009. WCSE'09. Second International Workshop on* (Vol. 2, pp. 371-375). IEEE.

[20] Fagin, R., Haas, L. M., Hernández, M., Miller, R. J., Popa, L., & Velegrakis, Y. (2009). Clio: Schema mapping creation and data exchange. In *Conceptual Modeling: Foundations and Applications* (pp. 198-236). Springer Berlin Heidelberg.

3.3 WSDarwin: A Web Application for the Support of REST Service Evolution

Fokaefs, M., Oprescu, M., Stroulia, E., 2015. WSDarwin: A Web Application for the Support of REST Service Evolution. In: IEEE International Conference on Software Engineering (ICSE 2015). IEEE.

Note: This paper has been submitted to the tool demo track of the International Conference on Software Engineering (ICSE 2015) pending review.

WSDarwin: A Web Application for the Support of REST Service Evolution

Marios Fokaefs, Mihai Oprescu and Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
Email: {fokaefs,oprescu,stroulia}@ualberta.ca

***Abstract*—REST has become a very popular architectural style for service-oriented systems primarily due to its ease of use and flexibility. However, their lightweight nature does not necessitate the use of systematic methods and tools. In this work, we argue that such tools can greatly facilitate complex engineering tasks such as service evolution and service discovery. We present the WSDarwin set of tools to generate WADL interfaces for REST services, compare service interfaces to identify differences between two versions and compare cross-vendor web services to facilitate service discovery and interoperability. Video URL: http://youtu.be/52CclMbJt6M**

I. Introduction

Service-oriented architectures have become the prevalent paradigm for development of modular software. REST services are particularly popular for their ease of use and flexibility, since they can be easily invoked through simple HTTP requests. Almost every language has libraries to allow HTTP requests and no additional software is required. REST services have also been gaining popularity over other service architectures like SOAP services. In fact, big companies like Amazon[1] and Google[2] have discontinued their SOAP APIs in favour of their RESTful counterparts. Amazon has also reported that 8% of the requests to their services comes through their REST API[3] and that querying the services using REST is 6 times faster than with SOAP[4].

The REST architectural style does not enforce any standardized specification. Although there exist standard interface formats like WADL[5] or WSDL 2.0[6] to specify REST services, providers keep publishing REST APIs as non-standardized, free-text HTML pages, which are easily understandable by humans but not as easily consumable by software. This lack of standardization allows for a variability in how client developers interpret the specification of a service and how the service's functions can be accessed. This flexibility deprives the developers from a systematic support towards developing and maintaining REST client applications. This lack of systematicity is a coin with two sides. On one hand, it facilitates the adaptation of REST services, but on the other, it makes handling the evolution of a service a difficult task for client developers.

In this work, we present a web application[7] that offers a variety of tools to support developers of REST services and service clients. The application works in a complementary and optional manner for both providers and clients, meaning that it doesn't add any overhead towards the production of the consumption of REST services, but rather facilitates both of these tasks. The application contributes three tools.

- The **WADL generator** automatically generates WADL interface specifications for REST services and using the `wadl2java` tool [8] provided by Oracle, it can also generate Java-based client proxies from the WADL files.
- The **service-interface comparator** automatically identifies differences between two versions of a service's WADL interface to study its evolution.
- The **cross-vendor service mapper** semi-automatically maps the response elements between two different services from the same domain to determine their substitutability.

The rest of the paper is organized as follows. In Section II, we present an overview of the application's tools and their functionality. In Section III we briefly review some existing tools relevant to the WSDarwin web application and Section IV concludes this work.

II. Web application overview

The WSDarwin web application is itself built based on a service-oriented architecture. In the back-end a web service contains the main functionality of the tools. A Javascript front-end receives the input from the users and visualizes the results of the web service. A PHP middleware listens for events from the front-end and invokes the web service with the appropriate parameters. Then, it parses the results and passes them to the front-end.

[1] http://www.stoneedge.net/forum/pop_printer_friendly.asp?TOPIC_ID=12687

[2] http://googlecode.blogspot.ca/2009/08/well-earned-retirement-for-soap-search.html

[3] http://www.theregister.co.uk/2006/04/29/oreilly_amazon/

[4] http://www.onlamp.com/pub/a/php/2003/10/30/amazon_rest.html

[5] http://www.w3.org/Submission/wadl/

[6] http://www.w3.org/TR/wsdl20/

[7] More information about the tool, along with a screencast and a link to the web application can be found at http://hypatia.cs.ualberta.ca/~fokaefs/index.php?option=com_content&view=article&id=60&Itemid=69

[8] https://wadl.java.net/wadl2java.html

A. WADL generator

Figure 1 shows the application's screen for the WADL generator. At the top of the screen, the user can provide as input one or more URLs that correspond to HTTP requests for the service, whose interface needs to be generated. The user needs to specify the type of the HTTP methods (GET, PUT, POST, DELETE) that corresponds to the URL and can add or remove URLs at will.

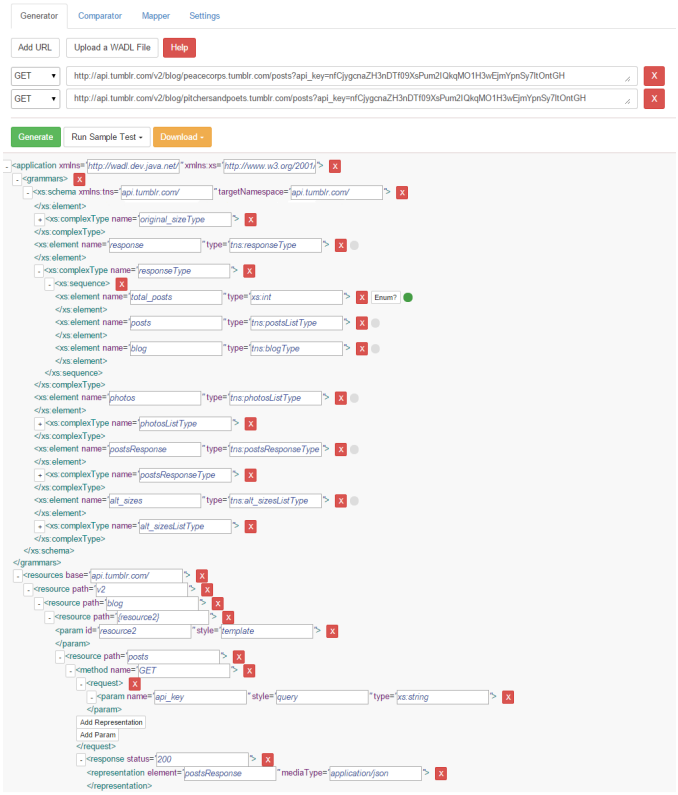


Fig. 1. The WADL generator.

The analysis of the input requests begins once the user hits the "Generate" button. The first step of the analysis is to parse every input URL. The URL string is split to determine the REST resources and their paths as shown in Figure 2. The first component corresponds to the `resources` path or, in other words, the service endpoint. The rest of the resources are nested within each other in a cascaded manner. If the URL contains a question mark ("?"), this implies a parameterized request and the parameters are separated by ampersands ("&"). The parameters come in pairs of name-value. The value of the parameter is used to determine the parameter's type. WSDarwin resolves types by using regular expressions on the value of the parameter. The tool can recognize a variety of primitive types including `integer`, `long`, `float`, `double`, `short`, `byte`, `date`, `dateTime`, `anyURI`, `email` and `string`. The analysis of an input request determines the resource structure of the REST service, the methods that are available and the input parameters of each method along with their type.

The second step of the WADL generation is to use the provided URL requests to invoke the service in order to get its response for the particular request. Once the response

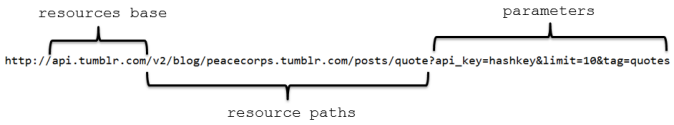


Fig. 2. Analysis of an input URL request.

is received, the WADL generator automatically determines whether it is in XML or in JSON format (the two most common response formats for REST services) and parses the file accordingly. The format is added as the `mediaType` attribute of the `representation` of the `response` element During the parsing of the response, its structure and the types of the data elements are determined, similarly to the input parameters. The result of this step is the inference of the XML schema of the WADL interface, which is added in the `grammars` element of the WADL.

After all provided URLs are analysed, the results are merged and a single WADL file is constructed and it is presented to the user with the appropriate XML annotation. The file is editable to allow the user to override the tool's decisions. The user can change the name or the type of a parameter and add or remove WADL elements. Eventually, the user can download the generated and edited WADL file locally. The user can also download a zipped file of a Java-based client proxy for the corresponding WADL file. The zipped file contains all the Java classes generated by the `wadl2java` tool provided by Oracle.

The batch processing of multiple URL requests gives to the WSDarwin web application three abilities; a) to resolve parameter types with certain confidence, b) to identify resources with variable identifiers and c) to identify enumerations.

The confidence with which a type is resolved depends on the number of provided URLs and how many of those this type was encountered in. For example, if out of the total 10 URLs that were analysed, the type for a parameter was found to be integer in 7 of them, then the tool reports the type as integer with confidence 70%. In case, multiple types are identified for the same parameter, the one with the highest confidence is reported or in case of ties, the largest type in terms of memory size (i.e., $string > double > float > long > integer$). The confidence is reported next to an element in the schema as a colour-coded circle; green for confidence between 100% and 75%, orange between 74% and 50%, yellow between 49% and 25% and red between 25% and 0%.

Resources with variable identifiers correspond to classes of entities, whose ID differs between instances. Examples of such resources include usernames and account IDs. The identification of variable IDs is performed during the analysis of the input URLs. If there is a path component that is different between the URLs, it is identified as a candidate. If this component is preceded by exactly the same path components and is succeeded by at least one common component, then it is identified as a resource with variable ID. In the WADL file, the actual ID of the resource is `resource` followed by the index of the corresponding path in the URLs, surrounded by curly brackets, e.g. {`resource2`}. A parameter with the same name is also added in the resource, whose `style` is `template`. This parameter informs the developer that a

concrete value needs to be provided by the client code to complete the construction of the URL request.

Enumerations are parameters, whose values are restricted within a particular set, e.g. countries, languages, gender and so on. The generator uses two heuristics to identify potential enumerations. First, it keeps track of the values for a parameter and if a value appears in more than one of the input URLs, this is an indication for an enumeration. Second, if the type of the parameter is `string` and the values are capitalized, which is a convention to specify enumerations, then this is a clearer indication. Nevertheless, the confidence for an enumeration depends on how complete the input URLs are, thus, the final decision lies with the user. The generator offers a button next to each potential enumeration, which refactors the element into a `xs:simpleType` with the enumeration values. This action is undoable.

B. Service-interface comparator

Figure 3 shows the screen of the application for the comparison of two versions of the same REST service. The comparator requires as input either two WADL files or a WADL file and a set of URLs or two sets of URLs corresponding to different versions of the service. In case a set of URLs is provided, the WADL generator is is invoked first.

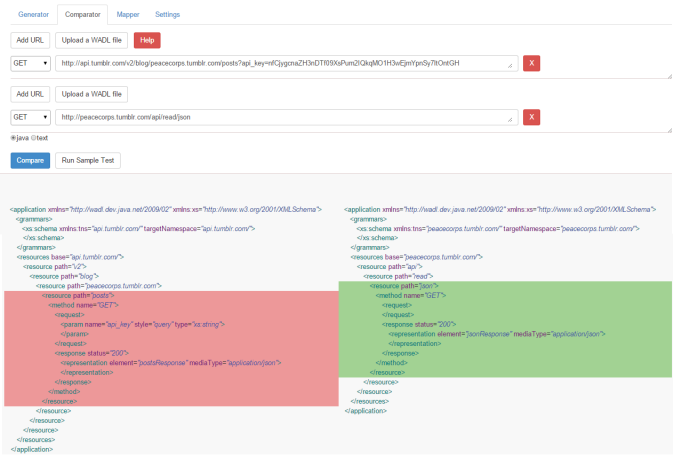


Fig. 3. The service interface comparison.

The comparison [1] first tries to map the service elements between the two versions. If the element has the same identifier between the two versions, then it is considered mapped. Alternatively, an element is considered mapped if it has retained the same structure between the two versions even if its identifier has changed.

After an element has been mapped, it is further compared to identify the particular changes between the two versions. WSDarwin can identify 5 types of changes: addition, deletion, move, change, and move and change. If an element cannot be mapped in the new version, it is considered deleted. If an element cannot be mapped in the old version, it is considered added. If an element's attribute or structure has changed, it is considered changed. To identify moves, the algorithm performs a second step. It compares all added elements with all deleted elements regardless of the level on which they were identified. If an added element can be mapped to a deleted element, then it is considered moved. If the element's attributes or structure have also changed, then it is considered moved and changed.

After the comparison, the results are presented to the user. The two WADL files are presented side-by-side with the changes clearly marked in the two files. Added elements are annotated with green on the right-hand file, deleted elements with red on the left-hand file, changed elements with orange, moved elements with blue and moved and changed elements with purple in both files.

C. Cross-vendor service mapper

Figure 4 shows the application's screen for the cross-vendor service comparison to map the elements between two different services from the same domain. The mapper requires as input one URL request from each service, since the mapping is performed on a single-method basis.

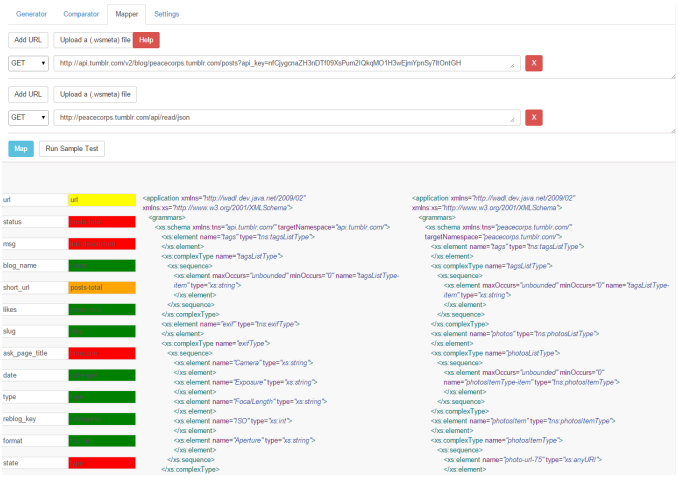


Fig. 4. The cross-vendor service comparison.

The goal of the mapper is to identify the compatibility between two services in terms of the data they return, in order to assess their substitutability and facilitate the client's migration from one service to the other [3]. Using the provided URLs, the tool invokes the services and obtains the corresponding responses. The responses are parsed and pairs of name-value are constructed for each response parameter. The types of the parameters are resolved as in the case of the WADL generation. The values of all the parameters from one file are compared to the values of the parameters of the other file assuming they are of the same type. Strings are compared based on their Levenshtein distance [2] and numbers based on their numeric distance. The mappings for each parameter are ranked from more similar to less similar. We allow many-to-many mappings under the assumptions that there can exist complex parameters whose data can correspond to more than one parameters from the other service.

After the comparison, the results are presented to the user. The two generated WADL files are shown side-by-side as in the case of the version comparison. On the left of the screen, there is an index with the identified mappings. The left column of the index correspond to the parameters of the first service. The right column shows the mapped elements from the second service. The colour of each parameter corresponds

to its similarity with the parameter from the left column; green corresponds to similarity between 100%-75%, yellow to 74%-50%, orange to 49%-25% and red to 24%-0%. The parameters from the index are clickable and the user is navigated to the corresponding WADL element in the file, which is highlighted in the file editor.

III. Relevant tools

In our previous work, we have already evaluated the version [1] and the cross-vendor [3] comparisons and have compared them with similar tools. In this section, we focus more on the generation of the WADL interface, since there are tools that perform this exact task. In this paper, we review two such tools namely soapUI[9] and RESTDescribe[10].

soapUI is a platform for web service development with support for both SOAP and REST services. The platform has functionality for schema inference and WADL generation from a URL request to an existing REST service. The tool invokes the service using the URL request and then parses the request and the response to construct the WADL interface, similarly to WSDarwin. However, unlike our web application, soapUI has a significant drawback; it can only process a single request at a time. This results in an incomplete interface with a single method that corresponds to the single request. WSDarwin allows for the processing of multiple requests in order to produce as a complete interface as possible for the whole service and not just for a single method. Furthermore, soapUI fails in other occasions. For example, it failed to resolve the type of parameter as boolean but instead it reported it as string since the values "true" or "false" appear as strings in the response. In other cases, the tool failed to produce a WADL file altogether. Finally, soapUI infers only the schema of the response, but not the rest of the WADL file (resources, methods), which is assumed that is already provided.

RESTDescribe is an online tool, which is the product of a Master's thesis [4]. The tool accepts one or more URL requests for a REST service in order to produce the corresponding WADL interface. The tool then presents the interface to the user, who can further edit it and save it locally. In case of types of parameters, RESTDescribe reports the level of confidence for the accuracy of the particular type. The WSDarwin generator is greatly inspired by RESTDescribe, but it goes one step further and it also invokes the service to analyse the responses as well. RESTDescribe creates WADL interfaces with only requests. Another difference is that when RESTDescribe encounters a purely RESTful request (i.e., without parameters), it does not assign an ID to the method element. Conversely, WSDarwin assigns to the method the same name as the resource that contains it (see Figure 1 for an example). This is not illegal since we are talking about different types of elements (`resource` and `method`). The anonymity of a WADL element can cause problems in the mapping step of any comparison task.

IV. Conclusion

In this work, we have presented the WSDarwin web application, a practical tool to support REST service developers and service client developers and especially to support the evolution of this style of services. The application offers a variety of tools in a lightweight and easy to use fashion; a tool to automatically generate the WADL interface for a REST service from a set of service requests, a tool to compare two versions of a WADL interface and a tool to compare the WADL interfaces from two different providers.

The tool makes two significant contributions. On one hand, it provides a set of tools, whose interface is in accordance with the general RESTful philosophy; a lightweight, easy to use application without the imposition of standards and standardized software artifacts. Although for SOAP services or other modular architectures, there is an amplitude of standards and tool support, REST insists on an ad hoc approach on development without too much tool support especially for complex tasks like service evolution. The WSDarwin web application attempts to provide this support without violating the general principles of the REST architectural style.

On the other hand, the proposed web application is a tool intended for everyone involved with the development and maintenance of REST services, be that providers, clients or researchers. Service providers can use the application to provide additional information to their clients about the service with minimal effort, thus not interfering with the actual development of the service. Service clients can obtain this information by themselves through this application even if it is not provided by the service vendors. Finally, researchers can use the tool to study the evolution of REST services and evaluate their methods on service discovery and interoperability.

[9] http://www.soapui.org/

[10] http://tomayac.com/rest-describe/latest/RestDescribe.html

Acknowledgments

The authors would like to acknowledge the generous support of NSERC, iCORE, and IBM. They would also like to thank Lukas Ziegler for his contribution to the development of the WADL generator.

References

[1] M. Fokaefs and E. Stroulia, "Wsdarwin: Studying the evolution of web service systems," in *Advanced Web Services*. Springer New York, 2014, pp. 199–223.

[2] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, February 1966.

[3] B. Bazelli, M. Fokaefs, and E. Stroulia, "Mapping the responses of restful services based on their values," in *Web Systems Evolution (WSE), 2013 15th IEEE International Symposium on*. IEEE, 2013, pp. 15–24.

[4] T. Steiner, "Automatic multi language program library generation for rest apis," Master's thesis, Institute for Algorithms and Cognitive Systems, University of Karlsruhe, 2007.

Chapter 4

Service Evolution Economics

Service-oriented architectures have gained considerable popularity and have now become the prevalent architectural paradigm for the development of modular and distributed systems. Unlike other more traditional modular systems, e.g. software libraries, service systems have a set of unique properties. First, the components of the system may lie outside the domain and control of a single entity. Second, web services are published through a service interface, which serves as a usage contract for the client, and very little information is exchanged between the provider and the client other than that, which is necessary for the service system to function. Third, as web architectures, service-oriented architectures require the existence of a continuous and live web connection between the web service and the client application. In the event of evolution, and especially in the case of an incompatible change, which will break the service interface, *i.e.,* the agreed contract between the provider and the client, the developers of the various components of the service will have to consider the technical implications of these changes and take action, *i.e.,* identify the nature of the change, create adapters for the new version and recompile client proxies. Given that web services are also business software components, since they were introduced to expose expertise or proprietary data, these technical considerations have direct and parallel economic and business implications. The evolution of a web service will create direct and indirect (*externalities*) costs to providers and clients, it will affect the price of the service and of its competitive software and it will potentially reshape the

market.

An externality (Laffont, 2008) is an indirect effect of production or consumption activity that does not work through the pricing system. The effect is indirect in the sense that it impacts entities in the ecosystem other than the originator of the activity or decision. In the context of service ecosystems, the externalities for a client may be the adaptation costs caused by the evolution of the service that the client is currently using, a better opportunity offered by a competitive provider or the change in service prices. For a provider, an externality may be the increased competition caused by the improvement of a competitive service or the entrance of new competitors, new or lost opportunities caused by a change in the market (a client switching to a competitive provider or new clients entering the market). By taking into account the externalities of their decision, the participants of the ecosystem may be lead to better outcomes, which may not seem optimal at first glance. This coincides with the *"The General Theory of Second Best"* (Lipsey and Lancaster, 1956), which states that it is possible to move one or more variables away from their optimal value, in order to obtain a more efficient outcome. In the context of service ecosystems, this efficient outcome may also correspond to a more socially efficient outcome. This last statement leads to another relevant theorem, the Coase theorem (Coase, 1960). According to this theorem, an efficient outcome can be achieved through negotiations and further payments between the involved parties under certain conditions (the parties act rationally, transactions costs are minimal, and property rights are well-defined). In the context of service ecosystems, the Coase theorem applies when providers support their clients during the evolution of the service and the clients stay with the same providers, for their own benefit, creating a stable environment characterized by strong business relationships.

Therefore, the decision around the evolution of a web service is a complex socio-technical task with economic implications. In my work, I propose that service evolution should be studied as such a problem and that game theory can be used as a tool to capture the complicated interactions between providers,their competitors and their clients. Initially, I study service evolution

within the context of a very simple ecosystem between one provider and one client to show that the economic implications of software evolution in service systems may be exaggerated compared to other software architectures and to demonstrate the use of game theory as a tool to make decisions about this issue (Fokaefs et al., 2013). I proved mathematically that the price of the new version is directly related to the client's adaptation costs, and therefore how externalities can be incorporated in the pricing system. I also showed that the combination of actions that maximizes the financial benefits for the provider and the client is the one that also maximizes the social welfare, *i.e.,* the cumulative benefits (economic and technical) of all the participants, which is for the provider to evolve and support the client during the adaptation process and for the client to stay with the current provider and adapt.

Using this mathematical proof, I create a simple tool in the form of a decision tree to guide the provider towards this particular decision that will maximize the ecosystem's welfare (Fokaefs and Stroulia, 2013a). The provider can calculate the economic parameters of the service evolution and the decision tool will provide the best possible decision. The tool can also guide the provider to shape some of the parameters, e.g. the nature of the change and the price of the new version, in order to make the optimal decision according to the mathematical proof.

Motivated by the last statement about the decision variables of the problem, *i.e.,* the evolution effort and the price, I developed an economic model consisting of a set of functions to calculate the economic parameters of the problem, including costs and values of software, and a set of optimizations to estimate the best possible value for the decisions variables. This model is developed within an extended software ecosystem of a competitive market with multiple competitive providers and multiple clients. The model was built under certain assumptions to support the understandability of the model, without reducing the importance of considering externalities and the realism and practicality of the problem. The most important assumption is that there is considerable information flow between providers and clients. This is reinforced by another assumption; that the model is concerned with paid services.

The monetary incentives motivate providers and clients to acquire more information from their counterparts within the ecosystem including information about requirements, how services are used and what clients expect to gain from using a service. This information can help us estimate values for (evolution/adaptation) costs, expected returns, which are required as input to the model. In Section 4.3, I provide more detailed explanations about the various assumptions around the construction of the model.

Within the context of this expanded ecosystem, I redefine the game model, which now becomes a two-stage game. In the first stage, the providers decide simultaneously whether to evolve their respective services and how (support clients or not) and in the second stage, clients react to the providers' decisions by choosing whose provider's the service they will use after one or more of those has evolved. The solution of the game (*i.e.,* the identification of a Nash equilibrium), which also constitutes the final decision for the participants, consists of the evolution strategy of each of the providers and the resulting market decision after the clients' actions. I demonstrate the use of these models within the context of a realistically synthetic case study of cloud computing ecosystem using a combination of real and generated data. In this example, I also show some simple methods to find a Nash equilibrium even for larger problem spaces and also discuss the social and economic implications for some of the possible outcomes of the game.

4.1 Software Evolution in the Presence of Externalities: A Game-Theoretic Approach

Fokaefs, M., Stroulia, E., Messinger, P. R., 2013. Software Evolution in the Presence of Externalities: A Game-Theoretic Approach. In: Mistrik, I., Bahsoon, R., Kazman, R., Sullivan, K., Zhang, Y. (Eds.), Economics-Driven Software Architecture. Elsevier, Ch. 11, pp. 243-258.

Software Evolution in the Presence of Externalities: A Game-Theoretic Approach

Marios Fokaefs[a], Eleni Stroulia[a], Paul R. Messinger[b]

[a]*Department of Computing Science, University of Alberta, Edmonton, AB, Canada*
[b]*School of Business, University of Alberta, Edmonton, AB, Canada*

Abstract

The architecture of service-oriented systems is defined by the services involved and the network of their usage interdependencies. Changes in an individual service may lead to the evolution of the overall architecture, as (a) different or new interactions may become possible and (b) existing partners may leave the network if their dependency needs are no longer fulfilled. Therefore studying the evolution of a service and the impact it may have on services and business partners that depend on it is essential to studying software-architecture evolution in the age of SOA. In such an environment with different and possibly independent parties, there may exist conflicting goals, e.g., one party may aim for evolution while another may desire stability. In this work, we model the interactions and decision-making process during the evolution of a system using a game-theoretic approach and we explore how variations in the dependencies and the information flow between the service provider and the clients impact the provider's decision-making process regarding the evolution of the service.

Keywords: software evolution, software engineering economics, service-oriented architectures, game theory, externalities

1. Introduction

Software evolves over time to include various enhancements, and these changes may increase its value. From a technical standpoint, evolution oc-

Email addresses: fokaefs@ualberta.ca (Marios Fokaefs), stroulia@ualberta.ca (Eleni Stroulia), paul.messinger@business.ualberta.ca (Paul R. Messinger)

curs to fix issues with the software behaviour, to extend its functionality with improved and new features, and to improve its design qualities such as maintainability, performance and security. Clearly these changes have financial implications for both revenues and costs. On the one hand, maintenance and evolution imply development costs, related to modifying and retesting the software, and a learning curve for the users who need to become aware of the changes and how they may benefit from them. On the other hand, evolution should add value for the software users, which may be associated with an increased usage fee.

Our understanding of how software is developed and consumed has been changing, as have the underlying technologies, architecture styles and lifecycle processes, and so has the process of software evolution and the economic concerns around it. In order to calculate revenues and costs when considering the evolution of software, one should be aware of the architecture of said software and the nature of the relationship between production and consumption.

From an architectural perspective, software was originally thought of as a "product" of a development team. In this scenario, the decision on when and how exactly to evolve the software has to consider the estimated cost of the change, in relation to the anticipated increase of revenue that the change may bring. Eventually, technologies like software frameworks and off-the-shelf components enabled the modularization of the software architecture and its development as an agglomeration of parts, developed and evolved by independent teams. In this scenario, the community recognized that, in addition to the trade-offs in the cost and value of the change of a component, the evolution of that component has to take into account the impact of the change on its clients, who have to decide whether to continue using the deprecated component version, or to "catch-up with the evolution" by adapting their own software. More recently, software is increasingly viewed as the composition and orchestration of "services". In this scenario, when service providers consider evolving their offerings, they also have to consider the impact of the service evolution on the consumers of their services. The decision-making process is similar to the second scenario of component evolution, yet different in an important respect: old service versions typically cease to be available (it is too costly to support the operations of multiple versions at the same time), which makes the impact of the change to the client immediate and potentially severe.

Clearly, software evolution presents complex challenges for the software-

market participants, governed by several factors. First, the software production and consumption relations, enabled by software-reuse technologies, imply complex dependencies among various development teams. Second, the various parties may have different, even competing, goals during the software-evolution process; for example, while providers envision extensions to potentially attract new clients, existing clients may prefer stability. Third, the various parties may have different levels of technical adeptness and knowledge; while component providers have deep knowledge of their implementation and can easily plan its evolution, the component consumers may find it difficult to understand the change and to trace its impact throughout their own software. And fourth, economic considerations, particularly pricing, can influence the extent to which the parties involved want to develop, acquire, maintain, and utilize the software.

In the presence of such complex relationships between the various parties, the reactions of one party to the decisions of another may affect the outcome of the evolution process. When one party makes a decision and some of the associated costs or benefits are borne by another party, such costs or benefits are known in the economic literature as *externalities* (Laffont, 2008). Applied in the context of software evolution, providers' decisions (e.g., evolution) often give rise to external effects (i.e., externalities) borne by clients (e.g., adaptation costs). By their very definition, externalities on downstream users are typically not fully factored into the decisions taken by the providers in the process of software creation and enhancement. Yet the welfare of the ecosystem as a whole - including providers and consumers - depends on the actions of all the parties, and typically the presence of externalities leads to suboptimal outcomes for a system as a whole. We are interested in studying the extent to which such issues arise in the context of software evolution and identify possible remedies for this problem, including concerted action by providers and users.

In this paper, we study the software-evolution process from the perspectives of two parties simultaneously: the perspective of the provider, which is the party that develops and enhances the software, and the perspective of the client, which is the party that consumes the software and adapts to its changes. We study the relationships of these two parties in the process of software evolution, in the context of three different software-architecture settings. We view each setting as a game, where the two parties take actions that affect the outcomes for both parties, and, for each game, we define the players' actions and calculate their payoff. Eventually, we focus on

service-oriented architecture, as the most prevalent one today, and we develop a theoretical framework to study the relationship between providers and clients and the constraints that it imposes in the ecosystem. We then examine whether there can exist a viable solution for all involved parties in a service system whereby software evolves in a beneficial way for all concerned. We describe the other architectural styles in order to stress the special conditions and challenges that web services may impose and in order to show that each newer architecture introduces new parameters and constraints in the software evolution problem. We conclude our work with propositions that summarize the results of our study and provide guidelines to providers and clients on how to make the best possible decision.

The rest of the document is outlined as follows. In Section 2, we provide the background of this work and we discuss the related literature. In Section 3, we describe in detail the various evolution scenarios and examine in detail the relationship between the provider and the client in each one of them. Finally, Section 4 concludes this work.

2. Background

This work touches upon the areas of software-engineering economics, software evolution and theories of software cost and value.

2.1. Software Engineering Economics

Software economics is a mature research area that deals with the ever challenging issue of valueing software and estimating the costs involved in its production. These issues may be exacerbated in the case of service systems, because of the peculiarities of such systems, some of which we have highlighted in this work. In their work, Boehm and Sullivan (Boehm, 1981; Boehm and Sullivan, 1999, 2000) outline these challenges and also how software-economics principles can be applied to improve software design, development and evolution. They define software engineering fundamentally as an activity of decision making over time with limited resources and usually in the face of significant uncertainties. *Uncertainties* pose a crucial challenge in software development that can lead to failure of systems. Uncertainties can arise from inaccurate estimation. For example, cost-estimation models developed for traditional development processes no longer apply to modern architectural styles and development processes, such as the ones around service-oriented software systems. Furthermore, due to lack or inadequacy of

economic and business information software projects may be at risk. Boehm and Sullivan also recognize the need of including the *value added* from any design or evolution decision. However, as they point out, usually there are no explicit links between technical issues and value creation. It is critical to understand that the value added by evolving a system does not only depend on technical success but also on market conditions. It is stressed that the cost should not be judged in isolation. As Parnas suggests "for a system to create value, the cost of an increment should be proportional to the benefits delivered" (Parnas, 1972). Finally, the authors claim that there is a need for not only better cost estimation models but also stronger techniques for analyzing benefits.

2.2. The provider-client relationship

The provider-client game as presented in this work is a clear example of an ecosystem where externalities exist. An externality is an indirect cost or benefit of consumption or production activity, i.e., effects on agents, other than the originator of such activity, which do not work through the price system (Laffont, 2008). External effects such as these can lead to suboptimal, or inefficient outcomes, for the system as a whole, whereby both parties by acting independently end up less well off than they could do if they coordinated their actions or if the decision maker (in this case the provider) took into account the external effects of any action.

The Coase theorem (Coase, 1960) argues that an efficient outcome can be achieved through negotiations and further payments between the involved parties under certain conditions (the parties act rationally, transactions costs are minimal, and property rights are well-defined). In this work, the last scenario, where the provider supports the client in the adaptation process, is an example of the Coase theorem.

The relationship between a producing party (provider) and a consuming party (client) is a prevalent concept in many economic and business fields. More specifically, in the field of operations management, the relationship between the provider and the client is a special case of a supply-chain relationship, where we have the provider of an input interacting with a firm using that input in the production process (Nagurney, 2006; Gokhan and Needy, 2010; Oliver and Webber, 1992). In the field of marketing, the relationship is referred to the channel of distribution (Choi, 1991; McGuire and Staelin, 2008). In both of these fields, there is an external relationship between the upstream supplier/provider and the downstream producer/client.

Hoffmann (2007) studies the interbusiness relationships as a portfolio of strategic alliances and how an evolving environment can affect these alliances. According to the author, there can exist three strategies in managing the portfolio and coping with a changing environment; (a) actively *shaping* the environmental development according to firm strategy, (b) *stabilizing* the environment in order to avoid organizational change, and (c) reactively *adapting* to the changing environment. In the context of our work, we can perceive the different strategies as being part of different business partners. For instance, the provider is the one that shapes the environment by evolving the software, the client is trying to catch-up with the evolved software in order to stabilize the environment and reach a previous point of balance and other providers are trying to adapt to the changed environment in order to stay in the competition.

2.3. The value and cost of software evolution

Software evolution has been extensively studied, both as a technical problem as well as a decision-making process. In this section, we review several works that touch upon various aspects of the software-evolution problem as described in our work.

In order to calculate the value that the provider expects to receive from the change, first the type of change has to be determined. According to Swanson (1976) changes in software systems can be either *perfective* (e.g., to add new features), *corrective* (e.g., to fix a bug), *adaptive* (e.g., to migrate to a new language) or *preventive* (e.g., refactorings). For each of these types of changes the value for the provider depends on different factors. For example, the value from fixing a bug can depend on the popularity of the bug (i.e., how many developers follow its updates), the importance of the code where the bug was found, its severity and so on. To calculate the value from adding new features, Tansey (2008) used financing and accounting measures, namely the Net Present Value index (NPV) along with software metrics to calculate cost and effort and projected the evolution of the system in the future in order to select the most profitable scenario. Finally, to calculate the value of preventive changes a lot of works in the field of software maintenance have used traditional software metrics to calculate the improvement in design quality, maintainability and understandability of the code.

Ozkaya et al. (2007) propose a quality-guided model to evaluate architectural patterns and design decisions to support the decision process of software designers and architects. They employ real-options analysis to identify the

best available design decision. In their analysis, they take into account and study the effect of the decision on a set of quality properties (rather than just one).

Many methods have been proposed to estimate the implementation cost of changing software. One of the most popular ones is COCOMO II (Boehm et al., 2000). This model calculates cost as the programmatic effort required to change the software in terms of source lines of code or function points. An issue with this model is that it requires knowledge about the system's source code. When the source code is not available, for example in the third scenario we describe, the provider cannot predict the adaptation cost for the client and therefore cannot make an informed decision. This issue is mitigated by an extension of COCOMO II, called COCOTS, which calculates costs when the system is using COTS. In particular, a sub-model of COCOTS, the *volatility* model, calculates the costs to adapt to changed COTS, when the source code is unavailable. However, this approach requires knowledge about the source code of the client applications, which does not facilitate the provider's decision-making process.

Srivastava and Sorenson (2010) propose a method to select between functionally equivalent services based on "quality of service" (QoS) properties. They study how the clients value the quality of the service and how they would react in case of a change in the QoS properties. These QoS properties are usually contained in the Service Level Agreement (SLA) which is an artifact that can be used by the provider to calculate variables concerning the value of service such as. Finally, the authors also argue (but not further investigate) that the client's reaction to a change in QoS properties also depend on the price fluctuation for the service.

Having reviewed the aforementioned works (concerning both value and cost of evolution) we recognize the need for a model of the software-evolution process, in the context of an ecosystem, rather than just as a process carried out by the service provider as an independent entity. This model should include all the relevant costs and benefits for providers and clients alike. As we have seen in this work, certain decisions, which might look optimal for one party, might not be optimal for ecosystem as a whole, and thus lead to inferior outcomes for the individual parties.

3. The software evolution game

Adopting a game-theory perspective to analyze the provider-consumer relationship in the context of software evolution, we distinguish three different scenarios in the software-evolution game. The distinction is based on the architecture and underlying technologies of the evolving software system, which, to an important extent, dictates how the software is delivered and shapes the relationship between the developing party (provider) and the consuming party (client).

- *Software is a monolithic product, produced by a single independent entity (individual or organization).* The decision on whether or not to change it is made by the producing organization with full knowledge of the complete software; thus, the decision makers can assess the full scope of the change and estimate its cost. At the same time, the software organization can also estimate the additional value that the change will embed in the software, and develop a plan for its eventual monetization.

- *Software is built and delivered as a module, either an off-the-shelf component (COTS) or an extensible framework; the client application is built using these modules.* The various constituent parts are owned and are being developed by different and independent organizations. Therefore, the evolution of the reusable modules may impact their clients if they decide to migrate to the newer version. However, the clients may choose to keep using the older module versions, which remain typically available but may no longer be supported or maintained by the provider.

- *Software is built as a service; the client application is built by composing a number of services.* Software is still modular but the fundamental distinction between this scenario and the one above is that, in this case, the reusable components are available online and accessible at run time; the client does not own "copies of earlier service versions" but rather the right to use/invoke the services, which are deployed by the provider. Given this type of tight run-time relationship between the provider and the client of reusable software components, the decision-making process around service evolution occurs in the presence of externalities. Although externalities exist in both the previous scenario and this one,

in the latter case they are more pronounced. Older versions of services cannot be available to the client, if they are not supported by the provider.

In fact, Kaminski et al. (2006) argue that backwards compatibility should be offered when evolving a web service, but at least until support for the older version is formally withdrawn. Practically this means that there should be a grace period during which the clients should make efforts to migrate to the newer version. Consider for example Twitter [1] that released API v1.1 in September 2012 as a replacement for the old v1, which is supposed to be completely retired in March 2013. At this point, the provider will cease to offer the older version since there are costs associated with the maintenance of multiple versions simultaneously, and the clients will have to decide whether they will migrate to the new version or they will have to find a more suitable alternative.

Another important aspect of service-oriented architectures that is different from other architectures is the relationship between providers and clients. Services are conceived to implement (or support) high-level business offerings, and influence business interactions at a higher level than interdependencies between typical software components and libraries. After all, some of the most popular examples that are used to explain service systems (e.g. loan approval, product orders etc.) describe eponymous business transactions were some data needs to persist in the system, unlike the usually anonymous interactions in modular systems. Therefore, the nature of these transactions impose a stronger type of relationship between the provider and the client: that of business partners. In this scenario, the provider and the client may act completely independently, but, because of this business partnership, the provider may choose to support the client in the adaptation process, thus internalizing some of the client's cost for migrating to the new service version. This tighter relationship between the provider and the client and the motivation of the former to support the latter can also be explained by Williamson's transaction cost approach. In cases of tight relationship between two parties, where the cost of transactions between the parties is high, one party may opt to include the

[1] `https://dev.twitter.com/blog/planning-for-api-v1-retirement` (last accessed 15 February 2013

other in what is called the "efficient boundary" of the organization thus internalizing the transaction costs.

Although these settings differ from each other with respect to their complexity and the details of the relationship between the provider and the client, they share certain common aspects which, in this paper, we aim to capture in a coherent provider-client game-theoretic framework. In particular, we consider a situation in which a software provider is contemplating changing old software and making the new version available to the client. We write V_o^C as the value to the client of the old software (before the change), and we write p_o as the price the client pays for the old software.

In order to evolve the old software, the provider is assumed to incur a cost C_e (for such things as conceptual development and implementation). The software also requires investment of effort on the part of the client to adapt, assimilate, and use in the client's own systems and to teach the client team how to make use of the new system elements. We refer to these costs on the part of the client as adaptation costs, which we write as C_a.

In some cases, there is inherent value to the provider in improving the software, and this value, as perceived by the provider, we denote by V_e^P; for example, improvements may allow the software to more efficiently utilize the provider's resources. In all cases in our model, we assume that there is an increased value of the new version of the software to the client; this value, as perceived by the client, we denote by V_e^C. Naturally, V_e^P depends on the costs incurred to change the software, which we refer to as the evolution costs; the more effort the provider puts in the evolution, the greater the improvement that can be achieved. However, V_e^C does not depend on the adaptation costs. This is because the improvement is specific to the software and the provider's efforts; no matter how much effort the client puts in the adaptation, the improvement will be the same. The increase in the value for the client depends on the degree to which the client can benefit from the improvements that the provider built in the evolution.

Table 1 summarizes all the variables in our model. We observe that the notation distinguishes between two states for the software: *before* the evolution of the software and *after* the evolution. All the variables referring to the former state have the subscript o to denote that they are specific to the old version of the software, the variables referring to differentials (increase or decrease) have the subscript e and the variables referring to the new state have the subscript n. We distinguish between three types of variables:

Table 1: The variables and their definitions as used in the evolution model.

Variables	Definition
Before the evolution	
V_o^C	the value of the software before evolution (based on its features and qualities)
p_o	the price the client pays for the software (assumed to be either an one-time fixed price or based on a pay-per-use contract of some duration)
After the evolution	
C_e	the cost of the evolution process
C_{ai}	the cost of adaptation to the new version of the current provider's software for the client application
C_{aj}	the cost of migration to provider j's software for the client application
p_e^E	the price differential for the updated software when the provider just evolves
p_n^E	the new price for the updated software when the provider just evolves ($p_n^E = p_o + p_e^E$)
p_e^S	the price change for the updated software when the provider supports the client
p_n^S	the new price for the updated software when the provider supports the client ($p_n^S = p_o + p_e^S$)
p_j	the price the client pays for provider j's software
V_e^P	the change in the value of the updated software, as perceived by the provider
V_e^C	the change in the value of the updated software, as perceived by the client
V_n^C	the value of the new version of the software, as perceived by the client ($V_n^C = V_o^C + V_e^C$)
V_j^C	the the value of provider j's software, as perceived by the client
a	the subsidy rate, which determines what portion of the adaptation costs the provider will cover
b	the final portion of the adaptation costs that remains for the client after the provider's support

costs (denoted with the letter C), software prices (denoted with the letter p for price) and non-monetary benefits (denoted with the letter V for value). Finally, the variables that are specific to the "other" provider (to whom the client may choose to switch) have the subscript j.

3.1. Software as a monolithic product

In the case of a monolithic system, there is a single organization that designs, develops and maintains a single piece of software. As far as the delivery and usage of the system we can distinguish between two scenarios; either the firm develops the software for internal purposes or the firm sells the software as a shrink wrap to anonymous end users. In both cases, the development and the maintenance of the software, as well as all decisions concerning these activities are internal to the firm, which can make fully-informed decisions on this matters. The firm (i.e., the provider) then controls who is using the software (i.e., who the clients are) and can adapt the other processes or educate the personnel to use the new version of the software. All the costs incurred from the evolution process are internal for the provider of the software. In other words, the only cost present in this scenario is C_e. Furthermore, the benefit to be obtained through the evolution process is the system improvements with respect to its features and qualities, i.e., performance, design, efficiency etc., V_e^P. As a result the problem of software evolution becomes a linear optimization problem, where the provider has to choose the evolution scenario that maximizes the "profit":

$$\text{maximize } \Pi = V_e^P - C_e \tag{1}$$

$$\text{s.t. } V_e^P \geq 0 \tag{2}$$

$$C_e \geq 0 \tag{3}$$

In this scenario, there are no interactions external to the system. Both decision variables in this setting (V_e^P and C_e) are determined solely by the opportunities present in the environment, since there is no strategic interaction between players with divergent interests. As a result, the decision maker will have to, first, confirm that there exists an evolution scenario that will create a profit (i.e. $V_e^P - C_e > 0$) and, second, select the most profitable scenario.

Even if the software is a shrink wrap product sold to anonymous clients, these clients are not strategic players; instead they are an environmental

variable that feeds into the estimation of V_e^P (as a function of the proposed price per piece multiplied by the estimated number of customers)

The system might reach technological stagnation if no such evolution scenario exists. This may occur if the evolution cost becomes prohibitively high, due to the low technological adeptness of the provider. In such a situation, opportunities for evolution will not be pursued and the ecosystem will stagnate until the environment changes.

3.2. Software as a module

In this scenario, the decision maker has only partial control of the outcome, because the provider and the client are two independent entities that jointly influence the outcome. In this case, the client develops a software application that would consume the service and as a result the design, development and maintenance of the system is divided between an upstream (the provider) and a downstream developer (the client. Therefore, the costs and benefits are no longer internal to a single party and the information flow between them is somewhat reduced. Furthermore, the behaviour of each party and their reaction to the evolution event may affect the other, in the presence of externalities.

In the special case where the provider evolves the software but offers some backward compatibility so that old versions of the software are still available, the client applications continue to function as intended. We call this scenario "asynchronous evolution", in order to stress the fact that the client is not disrupted immediately after the evolution takes place. An example of such a scenario is programming libraries. This kind of software is usually offered as a compressed file, which the client can download locally and use. When the library changes, a new version is created and becomes available to the clients. However, the client may opt to keep using the old (now local) copy of the software, which, however, is no longer supported by the provider.

An implication of the fact that the software is now delivered as a module to an external client is that the module is offered for a price. This price can follow any pricing model, including a fixed one-time price, a pay-per-use pricing model or a subscription model. In any case, the price can be a function of time or instances (i.e., number of users, number of requests etc). We do not make any particular assumptions on how the price is calculated, but we consider as an input to our model.

Let us formulate this scenario as a game. On one hand, the provider has two choices: either to retain the status quo of the software and make no

changes (SQ), or to evolve the software (E). On the other hand, the client can continue to use the old version (O) or migrate to the new one (N). Figure 1 presents the offline evolution game in its extensive form.

If the client stays with the old version, then the provider still may perceive benefits from the new software, V_e^P (due, for example, to greater ease of maintaining the software), and the provider, of course, still incurs the cost of software change, C_e. But the provider does not receive revenue p_e^E for the new software, and the client does not pay for the new software. In contrast, if the client migrates to the new software, the provider does receive this revenue and the client has to pay for it. Furthermore, the client receives the net benefit of the new software, $V_e^C - C_{ai}$ (the value of the new software net of the adaptation costs).

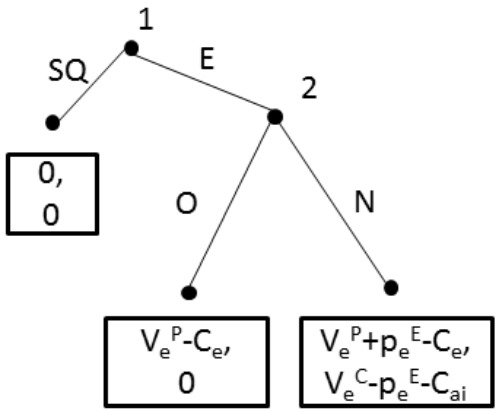


Figure 1: The provider-client game in the evolution of software as a module.

Table 2: best-response analysis for the evolution of software as a module game.

		When the client prefers...	
		$O \succ N$	$N \succ O$
Provider	$E \succ SQ$	1. $V_e^P > C_e$	2. $V_e^P + p_e^E > C_e$
Client	$N \succ O$	3. $V_e^C > p_e^E + C_{ai}$	

Using the best-response analysis as shown in Table 2, we find under what conditions each leaf of the tree is an optimal solution.The client will always

prefer to migrate to the new version ($N \succ O$), when the value of the new software for the client (V_e^C) exceeds the cost of its adaptation to the new version (C_{ai}) plus the associated price differential (p_e^E). On the other hand, if the client prefers to continue using the old version, the provider may still to choose to evolve if the value of the new software for the provider exceeds the evolution costs (C_e). If the client prefers to migrate to the new version, the provider will evolve the software if the value of the new software for the provider (V_e^P) plus the additional income p_e^E exceed the evolution costs.

Unlike the previous scenario, where the decision-making process involved environmental variables only, in this scenario the provider has a decision to make regarding the value of the price differential (p_e^E); by controlling this value, the provider has the power to avoid stagnant situations (where the software is not evolved).

In this scenario, the client will still work with the provider after the evolution, whether through the old software version or by migrating to the new one, thus guaranteeing an income for the provider. The problem for the provider becomes to decide on a price that will motivate the client to migrate to the new version, in order to obtain the additional income associated with the price increase of the new software version. In principle, the client prefers a lower price increase for a higher additional value. In fact, from inequalities 2 and 3 we have that $C_e - V_e^P < p_e^E < V_e^C - C_{ai}$, which defines the range for the price increase within which the client will select the new version and the provider will still make profit. Naturally, the provider will try to push towards the higher end of the price range, to increase the profit. Therefore, we have $p_e^{*E} = V_e^C - C_{ai}$, provided that $V_e^C - C_{ai} > C_e - V_e^P$. This means that the final decision is driven by client-oriented factors, even if there is no immediate positive outcome for the provider. Furthermore, the provider's net software development costs ($C_e - V_e^P$) must be covered by the price increase which will equal the net value of the evolution for the client. This amount for the price differential (p_e^{*E}) is the solution for this scenario.

3.3. Software as a service

In this scenario, the nature of the provider-client dependency is different from the one above. In the service-oriented paradigm, client applications are developed by consuming and composing services. Therefore, the provider and the client are highly coupled and if a change happens to the service the consuming party is affected. Unlike the previous scenario, in the evolution of software as a service, if the sum of adaptation costs and price increase is

too high, then it may be easier for the client to opt to abandon the provider. In this case, the provider not only tries to increase revenues, by evolving the software, but at the same time must take into consideration the possibility that current clients will abandon the new software.

The game corresponding to this scenario is shown in Figure 2. The provider has the same options as previously (SQ and E), but the client can now stay with the current provider and adapt to the new version (A) or leave the provider (L). We consider this setting to be a closed environment and as a result we do not examine how the client will choose the new provider, but we include in the model the parameters that are relevant to the other provider. These parameters, which will be noted with the subscript j (as opposed to the subscript i for the current provider), include the value the client will get from using the other provider's service (V_j^C), the price for provider j's service (p_j) and the costs for the client to adapt to provider j's service (C_{aj}).

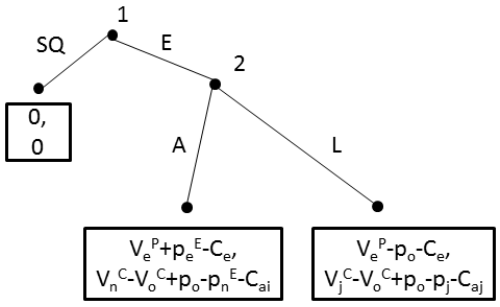


Figure 2: The provider-client game in the evolution of software as a service.

Table 3: best-response analysis for the evolution of software as a service game.

		When the client prefers...	
		$A \succ L$	$L \succ A$
Provider	$E \succ SQ$	1. $V_e^P + p_e^E > C_e$	2. $V_e^P > p_o + C_e$
Client	$A \succ L$	3. $V_n^C - V_j^C > p_n^E - p_j + C_{ai} - C_{aj}$	

The best-response analysis for this game is shown in Table 3. The client will always prefer to adapt to the new version ($A \succ L$) when the total value of the service (including the original value of the service (V_o^C) and the value of the new software for the client (V_e^C)) exceeds the adaptation costs plus the

price differential p_e^E and the original price p_o. In this case, the provider will prefer to evolve the service if the value of the new software for the provider plus the extra income p_e^E exceed the evolution costs of the provider.

On the other hand, if the client opts to leave the current provider, the latter will always prefer to evolve, if the value of the new software for the provider is greater than the evolution costs plus the current revenue from the client (p_o) that the provider will lose in this case.

Unlike the previous scenario, this is not a simple pricing problem, because now the provider has to consider the competition. In order for the evolution to be a profitable option, the provider will have to either increase their own benefits or make an effort to retain the client. The former can be achieved by

(a) increasing the value of the service (V_e^P),
(b) increasing the price for the new version of the service (p_n^E) and/or
(c) decreasing the evolution costs.

The latter option can be achieved by

(i) increasing the value of the service for the client (V_n^C) by offering additional features or better quality,
(ii) offering the service at a more competitive price (p_n^E) than the other provider and/or
(iii) decreasing the adaptation costs of the client.

Since improving the value of the service will require more effort from the provider, it becomes obvious that condition (c) is in conflict with conditions (a) and (i). Furthermore, conditions (b) and (ii) naturally contradict each other. Therefore, the only option that remains for the provider is to somehow assist the client in reducing the adaptation costs.

This can only be shown by condition 3 from Table 3. If we solve this condition for p_n^E, it will give us an upper bound for the price of the new version of the service $p_n^E < p_j + V_n^C - V_j^C + C_{aj} - C_{ai}$. Therefore, by increasing the value of the service or by decreasing the adaptation costs, the provider can increase the price of the service without losing any competitive advantage. In the next scenario, we are going to discuss how the provider can efficiently facilitate the client with the adaptation costs in order to increase the price of the new service.

3.3.1. Evolution with support.

The two previous scenarios may lead to suboptimal solutions (and probably stagnation) if the adaptation costs are so high that the net benefit of the client cannot cover the provider's potential loss (i.e., $C_e - V_e^P > 0$ and $V_e^C - C_a < C_e - V_e^P$). This can happen due to technological inadequacy on the client's part, or lack of communication and reduced information flow between the two parties. For this reason, in this scenario, we consider a particular type of information flow from the provider to the client, where the provider covers part of the adaptation costs by means of providing additional technical support to the client, to simplify its adaptation to the new version. The concept of technical assistance in the adaptation process has been a subject of extensive research and various methods have been proposed. Chow and Notkin (1996) have proposed that the provider give additional information about the evolution process so as to help the client in the adaptation process. On the other hand, Benatallah et al. (2005) propose a methodology to provide web service adapters on the provider side to make changes transparent to the client. Finally, Fokaefs and Stroulia (2012) propose an algorithm to support the automatic adaptation of applications on the client side.

By providing support to the client, the provider effectively "subsidizes" part of the adaptation cost. Assuming that the provider is more knowledgeable and technologically equipped with respect to the evolving service, then the provider can provide adapters with less effort than the client would need to create them. Therefore, if the provider produces adapters as a portion of the adaptation costs, say aC_a, where $0 < a \leq 1$, then instead of the remaining cost ($(1-a)C_a$), the client will bear a portion of the adaptation cost, say bC_a, such that $b < 1 - a$. Therefore, the provider can take advantage of this difference and charge a different price when supporting the client (p_n^S) than when just evolving the service (p_n^E). The corresponding game is similar to the previous scenario with the difference that the provider now has the additional option of supporting the client (S) as shown in Figure 3.

If we solve inequalities 7 and 8, we obtain an upper bound for the new price when the provider evolves with and without supporting the client adaptation. Naturally, the provider will push towards these upper bounds to maximize revenue and we can say that the two prices can marginally equal their upper bounds. Therefore, we have that $p_n^S = V_n^C - V_j^C + C_{aj} - bC_{ai} + p_j$ and $p_n^E = V_n^C - V_j^C + C_{aj} - C_{ai} + p_j$. If we subtract the second equation from the first, we have that $p_n^S - p_n^E = (1-b)C_{ai}$, which means that the

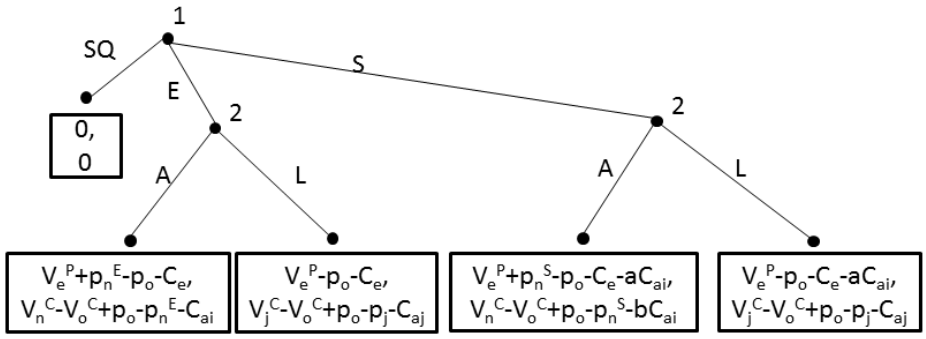


Figure 3: The provider-client game with support from the provider.

Table 4: best-response analysis for the evolution with support game.

		When the client prefers...	
		$A \succ L$	$L \succ A$
Provider	$E \succ SQ$	1. $V_e^P + p_n^E > C_e + p_o$	2. $V_e^P > C_e + p_o$
	$S \succ SQ$	3. $V_e^P + p_n^S > C_e + aC_{ai} + p_o$	4. $V_e^P > C_e + aC_{ai} + p_o$
	$S \succ E$	5. $aC_{ai} < p_n^S - p_n^E$	6. $-aC_{ai} > 0$ (ALWAYS FALSE)
		When the provider prefers...	
		$E \succ S$	$S \succ E$
Client	$A \succ L$	7. $V_n^C - V_j^C > p_n^E - p_j + C_{ai} - C_{aj}$	8. $V_n^C - V_j^C > p_n^S - p_j + bC_{ai} - C_{aj}$

provider can charge a higher price for the new version, while supporting the client. However, the difference between this higher price and the normal price of the new version should not be higher than the adaptation costs the client will save from the provider's support. Eventually, we have that $aC_{ai} < p_n^S - p_n^E < (1-b)C_{ai}$. Within this range the provider will make profit even after supporting the client and the client will be motivated to adapt to the new version rather than to switch to a different provider. If the client decides to abandon the current provider, the provider will not provide any support since this would be a frivolous and pointless action. This is reflected by condition 6 which is always false.

3.4. Propositions

Based on the analysis of the previous scenarios, we draw some conclusions about software evolution and the relationship between the provider and the client. These conclusions are summarized in the following propositions.

Proposition 1. ***The Nash equilibrium for the provider-client game is*** (S, A)**.**
Let us assume that the provider sets $p_n^S = (1-b)C_a + p_n^E$*. If the client already prefers to adapt when the provider evolves (i.e., condition 7 in Table 4 holds), then condition 8 (from the same table) also holds (it becomes the same as condition 7, in fact). Therefore, the client will always prefer to adapt. If condition 7 does not hold (i.e.,* $V_n^C - V_j^C < p_n^E - p_j + C_{ai} - C_{aj}$*), condition 8 becomes* $V_n^C - V_j^C > p_n^E - p_j + C_{ai} - C_{aj}$*; this means that* $V_n^C - V_j^C = p_n^E - p_j + C_{ai} - C_{aj}$*, which will make the client indifferent between adapting or leaving the provider.*
Similarly, if the provider prefers to evolve over supporting the client, condition 1 from Table 4 holds, and so does condition 2, which, in fact, becomes the same as condition 1. Therefore, the provider will never prefer to retain the status quo. If condition 1 does not hold (i.e., $V_e^P + p_n^E < C_e + p_o$*), condition 2 becomes* $V_e^P + p_n^E < C_e + p_o$*; this implies that* $V_e^P + p_n^E = C_e + p_o$*, which makes the client indifferent between retaining the status quo and evolve or support. Finally, we know that* $aC_{ai} < p_n^S - p_n^E < (1-b)C_{ai}$*, which means that condition 5 holds and the provider will always prefer to support than just evolve.*
Therefore, if the provider selects a slightly lower value for p_n^S*, then both players will strictly prefer the outcome* (S, A) *over any other outcome of the game.*

Proposition 2. ***Collaboration between the client and the provider can guarantee greater payoff for the ecosystem overall.***
In the presence of competition, the only feasible option for the provider in order to retain the current client is to provide technical support when evolving the software. Using knowledge about the software and its changes, the provider can be more efficient in covering part of the adaptation costs than the client; the savings that this efficiency makes possible can be translated in an increase to the price that the client may be able to pay for the new software version, which eventually implies increased income for the provider. If the provider simply evolves and the client adapts, the accumulative payoff of the

ecosystem will be $V_e^P + V_e^C - C_e - C_{ai}$*. If the provider supports and the client adapts, the total payoff will be* $V_e^P + V_e^C - C_e - aC_{ai} - bC_{ai}$ *which is greater than the previous payoff because* $C_{ai} > (a + b)C_{ai}$*.*

The provider's support towards alleviating the client's adaptation costs will result in the client's increased trust of the provider. Client support is a widespread concept in modern business. For example, the automotive industry has implemented for a long time the concept of "after-sales service" with great success. Eventually, client care may guarantee greater revenue for the provider through brand loyalty.

Trust and commitment have been a central theme in marketing research. Morgan and Hunt (1994) propose a model of relationship marketing where trust and commitment have a central role. In such a model, certain activities by the business partners such as sharing values and better communication may result in increasing trust which in turn will strengthen the relationship commitment. Eventually, the propensity of the business partners to abandon the relationship and the uncertainty of the environment is reduced. In a service-oriented system, where the provider assists the client during the evolution process, trust and commitment have a similar role. Information sharing is also discussed by Li et al. (2006) as a process that enables global optimization and strengthens the relationship between the producing party and the consuming party.

4. Conclusion

In this work, we studied the evolution of software in the context of a provider-client ecosystem. We identified a number of distinct software-architecture styles, and their implications on how the software is delivered by the provider and used by the client for the downstratem development of new software. We then examined a number of corresponding evolution scenarios and developed game-theoretic models of the decision-making processes of the involved parties, i.e., provider and client, in each of these scenarios. Our game-theoretic approach enables us to better model the concerns around software evolution in the presence of externalities. In this work, we argue that the evolution of modern software cannot be studied from a single entity's point of view. Externalities play a very important role and should be taken into account by the decision makers. We further argue that decisions should be made with the aim to optimize the welfare of the ecosystem as a whole,

and not driven simply by the interests of a single organization. Because of the relationship between the provider and the client, self-interested decisions will lead to sub-optimal, even undesirable, outcomes in the longer run. Eventually, we put forward two propositions when evolving a service-oriented software system.

1. The Nash equilibrium for the software evolution game is reached when the provider supports client in the adaptation process and the client stays with the current provider and adapts to the new version (S, A). This equilibrium promotes technological progress (since the software evolves and the new version is adopted) and collaboration between the parties that can control this technological progress.
2. This collaboration between the provider and the client can lead to larger payoff for the ecosystem as a whole, since the provider's support induces the client to adapt to the new version and enables the provider to receive a higher price for its software.

References

Benatallah, B., Casati, F., Grigori, D., Nezhad, H. R. M., Toumani, F., 2005. Developing adapters for web services integration. In: CAiSE. pp. 415–429.

Boehm, B., Sullivan, K., 1999. Software economics: status and prospects. Information and Software Technology 41 (14), 937–946.

Boehm, B. W., 1981. Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ.

Boehm, B. W., Clark, Horowitz, Brown, Reifer, Chulani, Madachy, R., Steece, B., 2000. Software Cost Estimation with Cocomo II with Cdrom, 1st Edition. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Boehm, B. W., Sullivan, K. J., 2000. Software economics: A roadmap. In: Proceedings of the Conference on the Future of Software Engineering. ACM Press, pp. 319–343.

Choi, S. C., 1991. Price Competition in a Channel Structure with a Common Retailer. Marketing Science 10 (4), 271–296.

Chow, K., Notkin, D., 1996. Semi-automatic update of applications in response to library changes. In: Software Maintenance 1996, Proceedings., International Conference on. IEEE, pp. 359–368.

Coase, R. H., 1960. The Problem of Social Cost. Journal of Law and Economics 3, 1–44.

Fokaefs, M., Stroulia, E., 2012. Wsdarwin: Automatic web service client adaptation. In: CASCON '12.

Gokhan, N. M., Needy, N., December 2010. Development of a simultaneous design for supply chain process for the optimization of the product design and supply chain configuration problem. 22 (4), 20–30.

Hoffmann, W. H., Aug. 2007. Strategies for managing a portfolio of alliances. Strategic Management Journal 28 (8), 827–856.

Kaminski, P., Litoiu, M., Müller, H., 2006. A design technique for evolving web services. In: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research. CASCON '06.

Laffont, J., 2008. externalities. In: Durlauf, S. N., Blume, L. E. (Eds.), The New Palgrave Dictionary of Economics. Palgrave Macmillan, Basingstoke.

Li, J., Sikora, R., Shaw, M. J., Woo Tan, G., Oct. 2006. A strategic analysis of inter organizational information sharing. Decision Support Systems 42 (1), 251–266.

McGuire, T. W., Staelin, R., Jan. 2008. An industry equilibrium analysis of downstream vertical integration. Marketing Science 27 (1), 115–130.

Morgan, R. M., Hunt, S. D., Jul. 1994. The Commitment-Trust Theory of Relationship Marketing. Journal of Marketing 58 (3), 20.

Nagurney, A., 2006. Supply Chain Network Economics: Dynamics of Prices, Flows, and Profits. Edward Elgar Publishing.

Oliver, R. K., Webber, M. D., 1992. Supply-chain management: logistics catches up with strategy. Logistics: The Strategic Issues.

Ozkaya, I., Kazman, R., Klein, M., May 2007. Quality-Attribute Based Economic Valuation of Architectural Patterns. In: 2007 First International Workshop on the Economics of Software and Computation. IEEE, pp. 5–5.

Parnas, D. L., Dec. 1972. On the criteria to be used in decomposing systems into modules. Commun. ACM 15 (12), 1053–1058.

Srivastava, A., Sorenson, P. G., 2010. Service selection based on customer rating of quality of service attributes. Web Services, IEEE International Conference on 0, 1–8.

Swanson, E. B., 1976. The dimensions of maintenance. In: Proceedings of the 2nd international conference on Software engineering. ICSE '76. IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 492–497.

Tansey, B., 2008. Valuing software services: The real options-based modularity analysis framework. University of Alberta (Canada).

4.2 WSDarwin: A Decision-Support Tool for Web-Service Evolution

Fokaefs, M., Stroulia, E., 2013a. Wsdarwin: A decision-support tool for web-service evolution. In: IEEE International Conference on Software Maintenance, Early Research Achievement (ICSM 2013 ERA). IEEE, pp. 444-447.

WSDarwin: A Decision-Support Tool for Web-Service Evolution

Marios Fokaefs and Eleni Stroulia
Department of Computing Science
University of Alberta
Edmonton, AB, Canada
Email: {fokaefs,stroulia}@ualberta.ca

***Abstract*—Service-oriented systems are fundamentally distributed in nature, relying on external services accessible through their public interfaces. Distributed ownership and lack of implementation transparency imply special challenges in the evolution of such systems. In order to alleviate the challenge faced by the consumers of their services, providers should, in principle, take into account the impact that service changes may have on the client applications, in addition to considering the potential benefits to be gained from the evolution of these services. In this paper, we present a decision tree to support the provider's service-evolution decision-making process. Using game theory, we construct the tree that makes explicit the value-cost trade-offs involved in considering the potential evolution of services.**

I. Introduction

The service-oriented system paradigm and associated technologies were conceived to support the reuse of functionality in the context of the development of large-scale distributed systems. Since services are consumed through standard public specifications (primarily in XML), service systems are technology agnostic and do not require any knowledge about implementation details of the services they rely upon. Although these properties have led to the easy development of component-based applications, they have also given rise to new challenges. One of them concerns the evolution of such systems. Due to the lack of implementation information between the two parties, i.e., the provider and the client, the latter is not in a position to reason about the changes of the service or the motivation behind the change. This can potentially increase the cost of adaptation for client applications and can potentially motivate the client to abandon the current provider.

When changing a service, a provider aims at maximizing some anticipated benefits, originating from the improvement of the service functionality or the extension of its features in order to acquire more revenue from clients, while, at the same time, minimizing the costs associated with actually implementing the change. These benefits and costs are direct and visible to the provider. On the other hand, there are indirect benefits and costs (also known in economic theory as *externalities* [1]) for the provider stemming from the client's reaction to the change. If the change improves the service and its quality, and meets better (or more of) the clients' needs, then they may become more committed to the service and they may be willing to pay more for it, thus generating additional revenue for the provider. On the other hand, if the clients' cost to adapt to the provider's evolved service far exceeds the benefits from the new version or there are other alternatives in the market that better suit their requirements, they may decide to switch to a competitor service, depriving the current provider from a source of income. The implication is that service providers, while being in principle self-interested, should also consider their clientèle, since they would risk losing clients and/or failing to attract new ones. When calculating the implications of a particular service change, the provider must also consider the properties of the ecosystem (providers-services-clients) and work towards the mutual benefit of all involved parties.

In this work, we extend our previous work [2], where we proposed a provider-client game to analyse their interactions during service evolution. In this paper, we propose a decision tree as a decision support tool, based on the provider-client game. We, first, revise the game and outline how it is changed from the previous version to allow for a more realistic ecosystem with potentially many providers and many clients. We also revise the best-response analysis to define the conditions, under which certain decisions are made and which are used as the decision nodes in the proposed tree. The resulting decision tree not only allows providers to make optimal decisions, but also to understand why these decisions are optimal for them and for the ecosystem as a whole. The tool is currently being built as part of WSDarwin, our web-service evolution platform [3].

The rest of the paper is organized as follows. Section II describes the proposed framework. In Section III, we provide an overview of this work's background by presenting a selection of related papers. Finally, Section IV concludes the paper.

II. The Decision Support Framework

A. The Provider-Client Game

In our previous work [2], we argued that the provider-client relationship involves conflicting interests and contradicting goals. For this reason, we proposed a provider-client game to capture this relationship. Table I shows the variables used in the game and their description. We denote provider specific variables with the superscript P and client-specific variables with the superscript C. We denote the current provider with the subscript i and the competitor with the subscript j. The differential values with subscript ei refer to the different value/price of the old version of the current service and the

new version. The differential values with subscript ej refer to the different value/price of the current service (old or new) and the competitive service.

TABLE I: The variables and their definitions as used in the provider-client game.

Variables	Definition
V_{ei}^{C}	the differential value of the service of the current provider i
p_{oi}	the original price the client pays for the service before the evolution
C_e	the cost of the evolution process
$C_{ai\|j}$	the cost of adaptation to the new version of the current provider's service i or the competitive service j for the client application
$p_{ei}^{E\|S}$	the price differential for the updated software when the provider just evolves (E) or when the provider also supports the client's adaptation (S)
p_{ej}	the price differential between the current provider and the competitor
V_{j}^{C}	the the value of the competitor provider j's software for the client
a	the subsidy rate, which determines what portion of the adaptation costs the provider will cover
b	the final portion of the adaptation costs that remains for the client after the provider's support

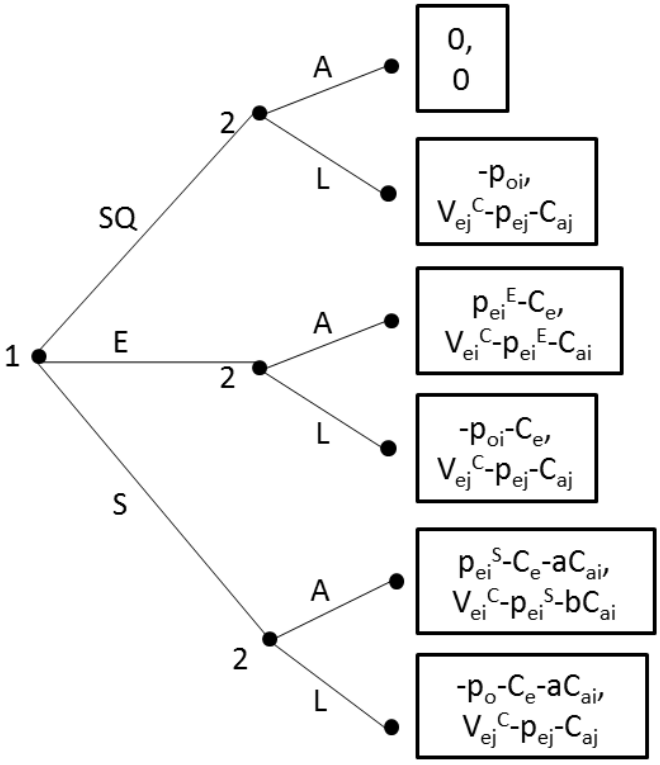


Fig. 1: The provider-client game in extensive form with the payoffs as the values for the leaves.

Figure 1 shows the provider-client game in its extensive form. The two values in the leaves of the tree correspond to the payoffs for the provider (first value) and the client (second value). In the game, the provider plays first and the possible actions are to:

1) **Maintain the status quo** (SQ) of the service and make no change. In this case, there is no difference in the provider's payoff, if the client decides to stay, but the provider loses the original income p_{oi}, if the client decides to leave.
2) **Evolve the service** (E), gain the increased revenues p_{ei}^{E}, if the client stays or lose the original income p_{oi}, if the client leaves and incur the evolution costs C_e.
3) **Support the client's adaptation (S)** and evolve the service. In this case, the provider gains the increased revenues p_{ei}^{S}, if the client stays or loses the original income p_{oi}, if the client leaves and incur the evolution C_e and part of the adaptation costs aC_{ai}.

Then the client responds and the possible actions are to:

1) **Adapt to the change (A)**, gain the differential of the value of the service from the old version to the new V_{ei}^{C} (or zero if the provider doesn't evolve the service) and incur the price differential (zero, if the provider doesn't evolve the service, p_{ei}^{E}, if the provider evolves or p_{ei}^{S}, if the provider evolves and supports) and any adaptation costs (zero, if the provider doesn't evolve the service, C_{ai}, if the provider evolves or bC_{ai}, if the provider evolves and supports, where $C_{ai} > bC_{ai}$).
2) **Leave the current provider (L)**, gain the differential of the value of service between the current provider and the competitor V_{ei}^{C} and incur the price differential p_{ej} and any adaptation costs C_{aj}.

The reason why we permit only two alternatives for the client is due to the nature of service systems. Unlike other modular software, for which clients can have a local copy of the software module and invoke on demand, in service systems, the web service has to be always online, since it is accessible over a network. Any change in the service may disrupt the proper function of the clients. Maintaining multiple versions of the service at the same time may prove very costly for the provider. In this case, the provider has two options. The first is to give to the clients a grace period before making the changes to the service, so that they have time to react properly to the change. An example of this case is Twitter [1], which released API v1.1 in September 2012 as a replacement for the old v1, which was completely retired in March 2013. At the end of the grace period, the clients will still have to decide whether they will adapt to the new version or switch providers. The second option is that the provider creates adapters that have the old version's interface but invoke the new version of the service. This idea was proposed and investigated by Kaminski et al. [4] and Fokaefs and Stroulia [3]. This option is in fact modelled by the support action of the provider in the proposed game.

A first difference between this game and its previous version [2] is that we no longer consider a value of the service for the provider and the income is only represented by the price.

[1] https://dev.twitter.com/blog/planning-for-api-v1-retirement (last accessed 17 June 2013

Another very important difference is that the client can now leave the current provider even if the latter retains the status quo. Assuming rationality from the client's part, if no provider evolves their services, then the client has no reason to switch providers. However, if one of them evolves the service, the client may opt to leave the current provider, if a competitor offers a better service. For simplicity purposes, we do not include the competitors as strategic players in the game, but we rather include the effect of their decisions in the calculations of the payoffs of the client.

Our analysis can be easily extended for multiple clients. We can safely argue that a client's decision does not depend on the other clients' decisions. Therefore, the outcomes and the payoffs for all players depend only on the provider's decisions and the competition. Therefore, we can run an instance of the game for each client and then aggregate the results and make the decisions that satisfies as many clients as possible.

B. Solution Concepts and the Decision Tree

Having defined the interactions between the provider and the client, we can now proceed to analyse the game using the backward induction algorithm and the method of best-response analysis, i.e., what action a player prefers given the preferences of the other players. Backward induction is a solution algorithm for sequential games, which first calculates the optimal decision for the player that plays last for each of the previous player's actions. Then these optimal actions are taken as a given for the previous player. The process is continued until we reach the player that plays first. The final path gives us the equilibrium of the game. The best-response analysis will give us the conditions under which a player has certain preferences and which we will use to construct the decision nodes of the decision tree.

Table II presents the best-response analysis for the provider-client game. As we can see from the table, conditions 2, 4 and 6 (i.e., when its more preferable for the client to leave the current provider) will never hold. This means that if the client leaves (i.e., conditions 7 and 8 do not hold), the provider will opt to retain the status quo of the service since this is the action that minimizes the losses for the provider. In an expanded ecosystem with many clients, the provider may balance the losses from clients that leave by attracting potentially new clients.

Figure 2 shows the final decision tree for the provider when considering the evolution of a service. The decision nodes are shown as rectangles with the number of the corresponding conditions. For each of the decisions we have two edges; one when the conditions is **T**rue in the left and another when the condition is **F**alse in the right. The end nodes of the tree representing the outcome of the decision process are shown as solid triangles with the final action for the provider.

As discussed above, all the paths for which the client leaves (i.e., $1(T) -> 5(T) \rightarrow 3(T) \rightarrow 8(F)$, $1(T) \rightarrow 5(F) \rightarrow 7(F)$ and $1(F) \rightarrow 5(T) \rightarrow 3(T) \rightarrow 8(F)$) lead the provider to retain the status quo of the service. The provider is led to the same outcome in two more paths ($1(F) \rightarrow 5(F)$ and $1(F) \rightarrow 5(T) \rightarrow 3(F)$, in which, independently of the client's decision, it is not in the interest of the provider to evolve since the costs exceed any potential income. For the other outcomes, one path leads to the evolution of the service ($1(T) \rightarrow 5(F) \rightarrow 7(T)$). The subpath $5(T) \rightarrow 3(T) \rightarrow 8(T)$ leads to the evolution of the service with support to the client regardless of whether condition 1 holds or not. In the leftmost subtree under the root, condition 3 can never be false, since, because conditions 1 and 5 are true, we have that $S \succ E \succ SQ$. Furthermore, under the same subtree, when condition 5 is false, we have that $E \succ SQ$ and $E \succ S$, which means that action E is a dominant strategy for the provider and we don't have to check condition 3.

TABLE II: Best-response analysis for the provider-client game.

		When the client prefers...	
		$A \succ L$	$L \succ A$
Provider	$E \succ SQ$	1. $p_{ei}^E > C_e$	2. $-C_e > 0$
	$S \succ SQ$	3. $p_{ei}^S > C_e + aC_{ai}$	4. $-C_e - aC_{ai} > 0$
	$S \succ E$	5. $aC_{ai} < p_{ei}^S - p_{ei}^E$	6. $-aC_{ai} > 0$
		When the provider prefers...	
		$E \succ S$	$S \succ E$
Client	$A \succ L$	7. $V_{ei}^C - V_{ej}^C > p_{ei}^E - p_{ej} + C_{ai} - C_{aj}$	8. $V_{ei}^C - V_{ej}^C > p_{ei}^S - p_{ej} + bC_{ai} - C_{aj}$

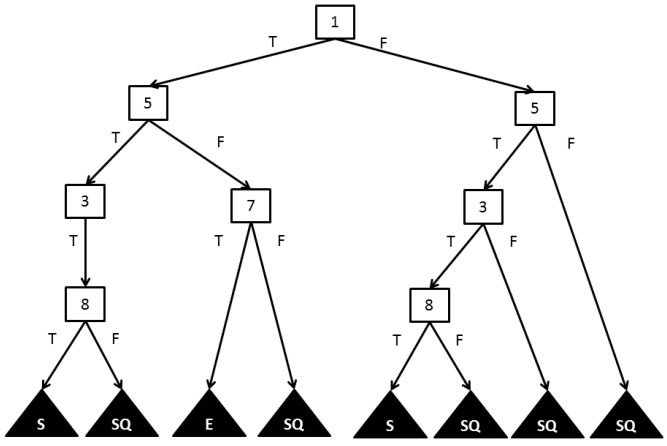


Fig. 2: The evolution decision tree for the service provider.

This decision tree is a simple tool that requires a considerate amount of information to be useful. However, this information is reasonably easy to obtain and in fact good enough estimates can be produced for all the variables required. First of all, there are variables that can be manipulated to guide the provider's decision towards a desired outcome. The provider can increase the value of the service at will and offer it at a competitive price, thus effectively overcoming competition. Furthermore, the provider can decide on the level of support

to the client to ease the adaptation process giving the client the incentive to stay. We can estimate the rest of the variables either by using formal models (i.e., for costs) or by surveying the environment of the service (i.e., for values and prices of competitive services).

III. Software Engineering Economics Background

Although software economics is a relatively mature field, analysing the cost and value of a particular software-engineering activity is an ever challenging problem. This problem is exacerbated in the case of service systems, because of the peculiarities of such systems, some of which we have highlighted in this work. In their work, Boehm and Sullivan [6], [7] outline these challenges and also how software-economics principles can be applied to improve software design, development and evolution.

Boehm and Sullivan define software engineering fundamentally as an activity of making decisions over time with limited resources, and usually in the face of significant uncertainties. *Uncertainties* pose a crucial challenge in software development that can lead to failure of systems. Uncertainties can arise from inaccurate estimation. For example, cost-estimation techniques and models that used to work for traditional development processes may not apply directly to modern architecture styles and development processes, such as web services. Furthermore, due to lack (or inadequacy) of economic and business information software projects may be at risk. They recognize the need of including the *value added* from any design or evolution decision. However, as they point out usually there are no explicit links between technical issues and value creation. It is critical to understand that the value added by evolving a system does not only depend on technical success but also on market conditions. It is stressed that the cost should not be judged in isolation. As Parnas suggests "for a system to create value, the cost of an increment should be proportional to the benefits delivered" [8]. Finally, the authors claim that there is a need for not only better cost estimation models but also stronger techniques for analysing benefits.

The provider-client game as presented in this work is a clear example of an ecosystem where externalities exist. An externality is an indirect cost or benefit of consumption or production activity, in other words, effects on agents other than the originator of such activity which do not work through the price system [1]. External effects such as these can lead to suboptimal, or inefficient outcomes, for the system as a whole, whereby both parties by acting independently end up less well off than they could do if they coordinated their actions or if the decision maker (in this case the provider) took into account the external effects of any action.

The Coase theorem [5] argues that an efficient outcome can be achieved through negotiations and further payments between the involved parties under certain conditions (the parties act rationally, transactions costs are minimal, and property rights are well-defined). In this work, the relationship between the provider and the client as we have described it through the game is an example of the Coase theorem.

IV. Conclusion and Future Plans

We argue that evolution in service-oriented architectures is a complicated task due to the complex dependencies between the participants of the service ecosystem. These dependencies are not only of technical nature but they also have economic and business implications. As a result strategic decisions concerning the evolution of a service should consider both technical and business dimensions of the ecosystem. In this paper, we showed how Game Theory can be used to model these dependencies and interactions between a service provider and a service client. We also proposed a decision tree as a tool to support the provider's decision-making process. This tool is based on estimates of economic parameters such as values, costs and prices and takes into account the clients' reactions to providers' decisions while at the same time considering the competition.

Although the model presented in this paper focuses on a single provider and a single client, it can be easily extended for more complicated ecosystems, while the general idea remains the same; service evolution in the presence of externalities. To this end, we plan to create decision trees for providers in different market settings: few clients with many providers (*oligopsony*), few providers with many clients (*oligopoly*), or many providers with many clients (*free/competitive market*). Another important part of our future plans is the validation of the decision support system. This can be achieved by simulating various evolution scenarios and testing our decision tree with (pseudo)randomly generated values. An alternative is an on-site evaluation of the decision tree by real service providers when considering the evolution of their service.

References

[1] J. Laffont, "externalities," in *The New Palgrave Dictionary of Economics*, S. N. Durlauf and L. E. Blume, Eds. Basingstoke: Palgrave Macmillan, 2008.

[2] M. Fokaefs, E. Stroulia, and P. R. Messinger, "Software Evolution in the Presence of Externalities: A Game-Theoretic Approach," in *Economics-Driven Software Architecture*, I. Mistrik, R. Bahsoon, R. Kazman, K. Sullivan, and Y. Zhang, Eds. Elsevier, 2013.

[3] M. Fokaefs and E. Stroulia, "Wsdarwin: Automatic web service client adaptation," in *CASCON '12*, 2012.

[4] P. Kaminski, M. Litoiu, and H. Müller, "A design technique for evolving web services," in *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research - CASCON '06*. New York, New York, USA: ACM Press, Oct. 2006, p. 23.

[5] R. H. Coase, "The Problem of Social Cost," *Journal of Law and Economics*, vol. 3, pp. 1–44, 1960.

[6] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice Hall, 1981.

[7] B. Boehm and K. Sullivan, "Software economics: status and prospects," *Information and Software Technology*, vol. 41, no. 14, pp. 937–946, 1999.

[8] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972.

4.3 Software Evolution in Web-Service Ecosystems: A Game-Theoretic Model

Fokaefs, M., Stroulia, E., 2015. Software Evolution in Web-Service Ecosystems: A Game-Theoretic Model. IEEE Transactions on Services Computing.

Note: This paper has been submitted to IEEE Transactions on Services Computing pending review.

Software Evolution in Web-Service Ecosystems: A Game-Theoretic Model

Marios Fokaefs, *Student Member, IEEE,* and Eleni Stroulia, *Member, IEEE*

Abstract—Service orientation is the prevalent paradigm for modular distributed systems, giving rise to service ecosystems defined by software dependencies, which, at the same time, carry business and economic implications. And as the software evolves, so do the business relationships among the ecosystem participants with corresponding economic impact. Therefore, a more comprehensive model of software evolution is necessary in this context, in order to support the decision-making processes of the ecosystem participants. In this work, we view the ecosystem as a market environment, with providers offering competing services and evolving these services to attract more clients by better satisfying their requirements. Based on an economic model for calculating the costs and values associated with service evolution, we develop a game-theoretic model to capture the interactions between providers and clients and support the providers' decision-making process. We demonstrate the use of our model with a realistic example of a cloud-services ecosystem.

Index Terms—web-service ecosystems, software evolution, game theory, software cost

1 Introduction

Web services are independently built and functionally autonomous pieces of software. Systems comprised of web services are fundamentally distributed in nature, with their constituent services typically provided by more than one organization. As a result, web-service systems create both technical and business dependencies between organizations, and they implicitly give rise to fairly tightly coupled business ecosystems.

In these ecosystems, the problem of software evolution is no longer simply technical. When a service changes, the client systems that use this service also need to adapt. In the presence of business relationships – client applications typically pay for the services they use – the provider's decision (on whether and how to evolve a service) and the client's decision (on whether to adapt to the evolved offering or switch to a competitor) have to take into account more information than what is conveyed by the public API of the service. As we have discussed in our previous work [1], the partners also have to consider the indirect effects, known as *externalities* [2], of their potential actions to the rest of the ecosystem. For example, when providers decide to evolve their service offerings, they have to consider the adaptation costs that the clients will have to bear and how these clients may react to this eventuality and also take into consideration the availability of similar and competitive services; similarly, when a client decides to abandon a provider after its service has evolved (potentially due to the high adaptation costs associated), the resulting loss of income is an externality for the provider in question.

The objective of our work is to develop a conceptual framework for reasoning about the evolution of service-oriented systems, recognizing the importance of economics in the decision-making process. This framework is an essential precondition for developing meaningful support for providers to consider whether and how to evolve their services, which is the specific technical objective of our work. Providers that offer similar services need to optimize their service-evolution strategy by balancing a trade-off. On one hand, they have to evolve their services so that they satisfy, to the best degree possible, the clients' set of requirements; on the other hand, they have to constrain the amount of effort (and corresponding development costs) they invest in this evolution, or else the clients' adaptation costs may be prohibitively high and may lead them to abandon the evolved service, at a loss for the provider. This paper makes three contributions towards modelling this complex phenomenon and reasoning about the providers' trade-off.

1) We have developed a **game-theoretic model** to capture the interests and relationships between competing providers and their clients accounting for the relevant externalities.
2) We have developed an **economic model** to calculate the relevant service-evolution parameters, including evolution and adaptation costs and corresponding service values. This model also includes algorithms to systematically calculate optimal values (or close approximations) for service prices and development effort.
3) Finally, we have developed a **decision-support method** to assist providers in making the best

• *M. Fokaefs and E. Stroulia are with the Department of Computing Science, University of Alberta, Edmonton, AB, Canada*
E-mail: {fokaefs,stroulia}@ualberta.ca

decision about service evolution. The method involves "solving the game" to identify the equilibrium that corresponds to the providers' evolution actions and effort, the prices for the new services' versions and the division of the clients among the providers.

The rest of this paper is organized as follows. In Section 2, we discuss the related literature that applies on our work. In Section 3 we define the service ecosystem we will be using in our analysis on service evolution. In Section 4, we analyse service evolution in a competitive market environment using game theory and in Section 5, we present the economic model to calculate the economic aspects of service evolution. In Section 6, we describe how we can solve the game to find an equilibrium. In Section 7, we describe how the models can be used in an artificial example, inspired by real web services. Finally, in Section 8 we discuss the assumptions of our models and how they affect the validity of our findings and Section 9 concludes our work.

2 Related Work

In this section, we provide a background on software engineering economics and discuss methods on estimating the cost and value of software evolution. Moreover, we discuss works that have studied the business and technical relationships that can be formed in a software ecosystem.

2.1 Software-Engineering Economics

Software economics is a mature research area that deals with the ever challenging issue of valuing software and estimating the costs involved in its production. These issues are exacerbated in the case of service systems, because of the peculiarities of such systems, some of which we are highlighting in our work.

In their work, Boehm and Sullivan [3], [4], [5] outline these challenges and also discuss how software-economics principles can be applied to improve software design, development and evolution. They define software engineering fundamentally as an activity of decision making over time with limited resources and usually in the face of significant uncertainties. *Uncertainties* pose a crucial challenge in software development that can lead to failure of systems. Uncertainties can arise from inaccurate estimation. For example, cost-estimation models developed for traditional development processes no longer apply to modern architectural styles and development processes, such as the ones around service-oriented software systems.

Boehm and Sullivan [5] put forward a utilitarian view for software evolution, according to which the system in order to create value for any involved party, it must create value for all whose contributions are critical to the project's success. Failure to satisfy any of those critical contributors will mean overall failure of the project thus failing to satisfy any of the involved parties. Through our work, we emphasize the importance of the utilitarian approach.

2.2 Values and Costs of Software Evolution

Software evolution has been extensively studied, both as a technical problem as well as a decision-making process. In this section, we review several works that touch upon various aspects of the software-evolution problem as described in our work.

The value that a change is expected to contribute to a software system and how it will be calculated depends on the nature of the change. To calculate the value from adding new features, Tansey [6] used financing and accounting measures, namely the Net Present Value index (NPV) along with software metrics to calculate cost and effort and projected the evolution of the system in the future in order to select the most profitable scenario. To calculate the value of changes to fix the design of the system previous software-maintenance research [7] has used traditional software metrics to calculate the improvement in design quality, maintainability and understandability of the code.

Ozkaya et al. [8] propose a quality-guided model to evaluate architectural patterns and design decisions to support the decision process of software designers and architects. They employ real-options analysis to identify the best available design decision. In their analysis, they take into account and study the effect of the decision on a set of quality properties (rather than just one).

Many methods have been proposed to estimate the implementation cost of changing software. One of the most popular ones is COCOMO II [9]. This model calculates cost as the programmatic effort required to change the software in terms of source lines of code or function points. An issue with this model is that it requires knowledge about the system's source code. When the source code is not available, as is the case for service-oriented systems, the provider cannot predict the adaptation cost for the client and therefore cannot make an informed decision. This issue is mitigated by an extension of COCOMO II, called COCOTS, which calculates costs when the system is using Commercial-off-the-shelf (COTS) software. In particular, a sub-model of COCOTS, the *volatility* model, calculates the costs to adapt to changed COTS, when the source code is unavailable. However, this approach requires knowledge about the source code of the client applications, which does not facilitate the provider's decision-making process.

Although these works use financial and economic methods, they focus mostly on the technical aspects of the problems and not so much on the social aspects, which are an essential part of economics.

However, they clearly point out the need for a model of the software-evolution process, in the context of an ecosystem, rather than just as a process carried out by the service provider as an independent entity. This model should include all the relevant costs and benefits for providers and clients alike. In our work, we attempt to show that certain decisions, which might look optimal for one party, might not be optimal for ecosystem as a whole, and thus lead to inferior outcomes for the individual parties.

Focusing on the price (and the value) of web services, Lyons et al. [10], [11] recognize the distinctive characteristics of web services as business artifacts and study various business models to deliver and price web services. These business models and the strategic decisions around them may dictate the relationship between providers and clients, the target market segments and competition. The authors discuss different pricing models for web services including the freemium model, pay-per-use and the ad-based model. In our work, we do not assume a particular pricing model, but we are concerned about the service price as the final income for the provider (and consequently the expense for the clients).

In this paper, we expand on our previous work [1], where we studied the evolution of service systems from a theoretical perspective focusing more on the dual relationship between a single provider and a single client. In this work, we propose a game based on a expanded ecosystem. We still have two general types of strategic players, namely providers and clients, but we consider more than one player of each type. The implication is that now we can capture the competition between providers to attract as many clients as possible by evolving their services to offer more functionality with better quality. Moreover, we allow clients more choices with respect to which provider's service they will use.

2.3 Web-Service Ecosystems

In this section, we selectively review previous research on ecosystems whose participants conduct financial transactions as they exchange goods and services. This review is by no means complete and it is meant to set the broad background of our framing of service-system evolution as a software AND business interaction.

The provider-client game as presented in this work is a clear example of an ecosystem where externalities exist. An externality is an indirect cost or benefit of consumption or production activity, i.e., effects on agents, other than the originator of such activity, which do not work through the price system [2]. External effects such as these can lead to suboptimal, or inefficient outcomes, for the system as a whole, whereby both parties by acting independently end up less well off than they could do if they coordinated their actions or if the decision maker (in this case the provider) took into account the external effects of any action.

The Coase theorem [12] argues that an efficient outcome can be achieved through negotiations and further payments between the involved parties under certain conditions (the parties act rationally, transactions costs are minimal, and property rights are well-defined). In our work, the evolution scenario, where the provider supports the client in the adaptation process, is an example of the Coase theorem.

Hoffmann [13] studies the inter-business relationships as a portfolio of strategic alliances and how an evolving environment can affect these alliances. According to the author, there can exist three strategies in managing the portfolio and coping with a changing environment; (a) actively *shaping* the environmental development according to firm strategy, (b) *stabilizing* the environment in order to avoid organizational change, and (c) reactively *adapting* to the changing environment. In the context of our work, we can perceive the different strategies in the activities of the different business partners involved. For instance, the provider is the one that shapes the environment by evolving the software, the client is trying to catch-up with the evolved software in order to stabilize the environment and reach a previous point of balance and other providers are trying to adapt to the changed environment in order to stay in the competition.

Barros et al. [14] and Barros and Dumas [15] describe how web-service ecosystems have risen to become the predominant model in software solutions. Traditionally centralized domains, such as business solutions, electronic shops and electronic auctions are now publishing pieces of their functionality to create web service ecosystems. In their work, the authors describe the ecosystem on a very high level and include various roles such as provider, broker, intermediary, client, end-user and so on. In our work, the scope of the ecosystem is narrower to only include providers and clients and to study a very specific event within this scope, namely the evolution of a service and how this may affect the ecosystem. The authors also point out the challenges that service evolution may impose especially from a behavioural and policy perspective. They also stress the need for tool support for this problem since *"maintaining adapters to deal with [the] multiplicity of interfaces can be costly and error-prone."*

Caswell et al. [16] describe an expanded service ecosystem, with multiple providers and clients, where any partner can be a provider and a client at the same time. This work studies how value is created and exchanged within the ecosystem and proposes business models to calculate this value based on intangible concepts such as the relationship between the partners and the client's satisfaction. In the proposed work, the scope of the ecosystem is much narrower and it examines the reaction of the ecosystem on very

TABLE 1
The model variables and their definitions. Units: pm = person-months, $ = dollars

Type of variable	Variables (units)	Definition	Values for example
Input	M^i (pm)	the amount of effort for a provider i to achieve 100% satisfaction of a client's x requirements in person-months	Table 4
Input	ER^x ($)	the expected return for a client x for using a service	Table 6
Input	$w^{i\|x}$ ($)	the wage (the cost for each person-month) for a provider i or a client x	Tables 5, 6
Input	m_o^i (pm)	the amount of effort already invested by provider i for the development of the service up to the point when evolution is considered	Table 4
Input/ Decision	$p_{o\|n}^i$ ($)	the price for the old version (o) or the new version (n) of a service i	Tables 5, 5
Computed	$m_{ai}^{i\|x}$ (pm)	the amount of effort for a client x to adapt to a new version of the service i (m_{ai}^x) or for a provider i to support the client in the adaptation process (m_{ai}^i) in person-months. It is assumed that the effort required to adapt or support is directly analogous to the size and the complexity of the change.	Tables 4, 7
Computed	V_i^x ($)	the value of the software of provider i for client x.	
Computed	C_e^i ($)	the cost associated with service evolution for provider i.	
Computed	$C_{ai}^{i\|x}$ ($)	the cost for a client to adapt to a new version of service i or for a provider i to support their clients.	
Decision	m_e^i (pm)	the amount of effort for a provider i to evolve a service.	Table 4
Decision	X_i	the clientèle of a provider i as a set of client indices	Table 5

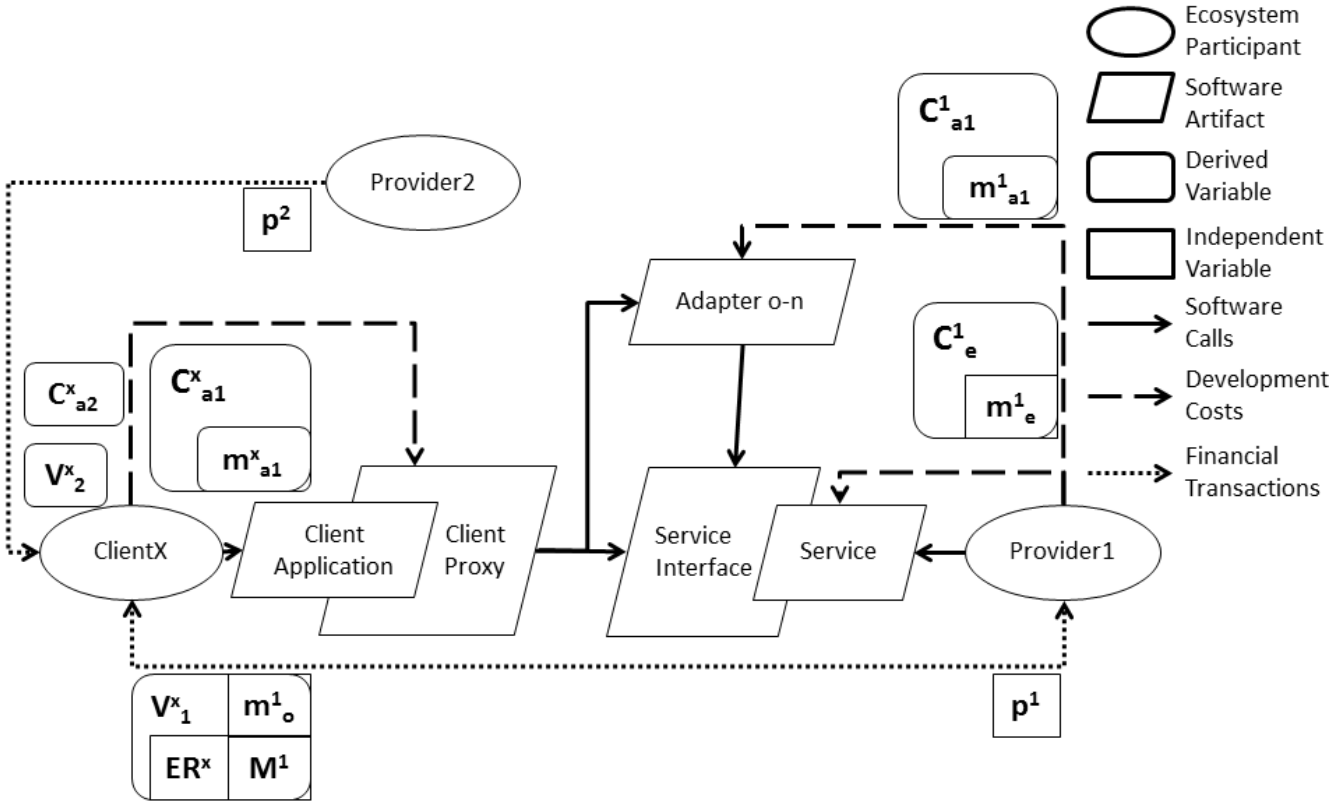


Fig. 1. Economic and technical interactions between providers and clients, during service evolution

specific events such as the evolution of a service.

3 Web-Service Ecosystems and Evolution

Recapping the problem at hand, web-service ecosystems consist of providers, who offer competitive services with similar features, and clients, who consume these services. Such ecosystems are fundamentally business environments and service evolution is an engineering task with direct economic implications. Clients may request changes and if their requirements are not satisfied they may switch to another offering. Therefore, they can push providers towards service evolution and technological advancement. Providers can use evolution as a marketing tool to create more competitive services and attract more clients. Evolution may also indirectly dictate pricing policies based on the new features of the service and the new division of the market. It is worth mentioning that in

this context the clients also represent business entities and not end-users. Therefore, we are talking about business-to-business relationships and transactions.

Let us first review the software transactions between the ecosystem participants. As Figure 1 shows, *ClientX* develops a *client application*, which consumes the *service* of *Provider1* through a public and programming-language-agnostic (usually in XML) *service interface*. In order to consume the service, a middleware is typically used to generate a *client proxy*, which is a representation of the service in the client application's language according to the service interface. In the event of service evolution, *Provider1* changes the service, which might affect the service interface, and consequently, *ClientX* will have to regenerate the client proxy and adapt to the new service interface. *Provider1* may assist *ClientX*'s adaptation process by providing additional evolution information [17] or *adapters* [18], [19] that will preserve the old service interface invoking the new functionality, thus minimizing the client's effort to adapt to the new version. A third alternative scenario is that the client chooses to switch to *Provider2*, in which case a new proxy needs to be generated and the client application needs to be migrated to the new service [20].

These software transactions create corresponding and parallel economic transactions between the participants. Table 1 lists all the variables used in our models to describe these economic transactions. The variables related to providers are indexed with $i, j \ldots$ and those related to clients with $x, y \ldots$. Special subscripts are also used to show the version of the service (o for old -before the evolution- and n for new -after the evolution-) or the development action (e for service evolution and a for client adaptation).

Focusing on the economic implications of the above software transactions, we observe that when *Provider1* offers a service to *ClientX*, the latter receives value V_1^x from using this service and the former receives monetary payment p^1 for providing the service. In the event of service evolution, the provider has a cost C_e^1 to change the service and the client has a corresponding cost C_{a1}^x to adapt to the new version. If the provider decides to support the client, then there is an additional support cost C_{a1}^1 for the provider which is subsidized from the total adaptation cost of the client. Finally, if the client decides to switch to *Provider2*, the client will have to cover the adaptation cost C_{a2}^x to migrate to the other service, and at the same time the client will receive a different value V_2^x and will have to pay a different price p^2.

The value of the service (further explained in Section 5.1) and the cost of a change (further explained in Section 5.2), either for evolution or for adaptation, directly depend on the features and the functionality of the service. These features are specified and implemented according to the clients' requirements. The requirements are determined by studying the environment and understanding its needs on a particular problem or they are explicitly solicited from the clients. These requirements are then formulated as use cases for the service [21], which in turn can be translated into Use Case Points (UCP) [22] or Function Points (FP) [23]. Using UCP as a metric to determine the size of a feature or of the software as a whole, we can estimate the effort required for the implementation of the features using COCOMO II measured in person-months (m_e or m_a for evolution and adaptation respectively). Thus, we showed that there is indirect but explicit correlation between features and effort. Having gathered all the requirements from all the different clients, the provider can now estimate the total effort required to satisfy the full set of requirements (M^1 for *Provider1* in Figure 3).

Although based on the same set of features, the effort is perceived differently by providers and clients in monetary terms. The value the clients receive from a service depends on their expected return ER^x. This return specifies how much the clients value each feature and its quality (functional and non-functional requirements) and it is different for each client. However, the return is not specific to particular service since the clients evaluate the desired features and not how those are implemented. The actual value that a client receives from a specific service (e.g. V_1^x for *Provider1*) depends on the effort that the provider of this service has invested towards satisfying the complete set of all its clients. The proposed model receives the expected return as an input and the calculation of which features each particular client requires and how much they value them is beyond the scope of this work. The monetary cost of the evolution (adaptation) effort can be estimated based on the monthly developer wage w as the nominal cost for a person-month.

The price p of a version and the total effort m_e invested by the provider in the evolution of a service vary based on the needs of the clients and the state of the ecosystem at a given time. As in any market environment, the provider will have to offer a service that best satisfies the clients' requirements at a competitive price. The estimation of price and effort, while taking into account the environment, is analysed in Section 6.

Finally, the decisions whether to evolve a service, support clients or switch providers are based on the economic parameters of the ecosystem calculated as previously, the interactions between the participants and the environment. This process is analysed in Sections 4 and 6.

4 The Providers-Clients Game

The fundamental premise of this work is that any decision about software evolution should be made taking into account its implications for the whole software ecosystem. More specifically, when making

decisions concerning evolution in a service ecosystem, participants will have to consider not only the development costs and the expected gains (i.e., increased income for providers and better services for clients), but also the externalities of these actions on other participants [1]. For example, the decision of a provider to evolve will create externalities (more competitive service and/or price) for their clients and their competitors. Similarly, a client's decision to switch providers will create externalities (loss/gain of revenue for the original/new provider) for the providers involved.

Since self-interested competing parties are considered in the ecosystem, the service environment can be modelled as a game and their interactions and behaviours are studied using game theory. The service-evolution game is a two-stage game. In the first stage, the providers decide independently from each other whether to maintain the status quo of their services, or evolve them, or evolve and support their clients. This stage is a simultaneous game and it can be represented as a normal form game.

In the second stage of the game, the clients react to the providers' collective actions represented by the Cartesian product of their action sets: $A^i \times A^j, \forall i, j \in I$. This stage of the game is solved for each client separately to find the final market division. The second stage of the game can be represented in an extensive form, where the provider group plays first and then the clients choose a service.

The service-evolution game is formally defined in Equation 1 in terms of the set of players $\mathcal{P}$, the set of actions for each player $\mathcal{A}$, the set of states of the game $\mathcal{S}$ as a combination of the players' actions and the set of utility functions for each player $\mathcal{U}$ per game state. The set of players for the service-evolution game consists of providers in set I and clients in set X. Each provider can either (a) retain the status quo of its service (SQ), (b) evolve its service (E), or (c) evolve its service and support its clients' adaptation (S). The actions of the clients correspond to the available services from set I; they can opt to adopt any one of them. It is assumed that each client uses a service and the question is whether the client will stay with the current service or switch to another provider in case the ecosystem changes (i.e., if any of the services has evolved). The states refer to the conditions of the ecosystem after a specific combination of player actions have been performed (i.e. at a leaf of the second stage of the game). Therefore, a state is characterized as said combination.

$$\begin{aligned}
&\mathcal{P} = \{I = \{i, j, \dots\}, X = \{x, y, \dots\}\} \\
&\mathcal{A} = \{A^i = \{SQ, E, S\}, A^x = \{I\}\}, && \forall i \in I, x \in X \\
&\mathcal{S} = (A^i)^n \times A^x, && \forall i \in I, x \in X, n = |I| \\
&\mathcal{U} = \{U^i(a_i, a_{-i}), U^x(i, s)\}, && \forall i \in I, x \in X, s \in \mathcal{S}
\end{aligned} \tag{1}$$

The utility of the client x (Equation 2) is a function of the chosen provider i and the current state of the game s. It depends on three factors; the value that the client receives from the service, the price of the service (new if the provider has evolved, and old otherwise) and any necessary adaptation costs.

$$U^x(i, s) = V_i^x(m_o^i + m_e^i) - p_{o|n}^i - C_{ai}^x(m_{ai}^x) \tag{2}$$

The utility of a provider i (Equation 3) is a function of the provider's action a, relative to all other providers' actions (a_{-i}). It is the sum of the revenue the provider receives from its clients, minus the costs of service evolution (if the provider has chosen to evolve its service), minus any support costs (if the provider has chosen to support its clients' adaptation).

$$\begin{aligned}
U^i(SQ, a_{-i}) &= |X_i| \cdot p_o^i \\
U^i(E, a_{-i}) &= |X_i| \cdot p_n^i - C_e^i(m_e^i) \\
U^i(S, a_{-i}) &= |X_i| \cdot p_n^i - C_e^i(m_e^i) - C_{ai}^i(m_{ai}^i)
\end{aligned} \tag{3}$$

TABLE 2
A two-provider game in normal form

	SQ2	E2	S2
SQ1	$U^1(SQ1, SQ2)$, $U^2(SQ2, SQ1)$	$U^1(SQ1, E2)$, $U^2(E2, SQ1)$	$U^1(SQ1, S2)$, $U^2(S2, SQ1)$
E1	$U^1(E1, SQ2)$, $U^2(SQ2, E1)$	$U^1(E1, E2)$, $U^2(E2, E1)$	$U^1(E1, S2)$, $U^2(S2, E1)$
S1	$U^1(S1, SQ2)$, $U^2(SQ2, S1)$	$U^1(S1, E2)$, $U^2(E2, S1)$	$U^1(S1, S2)$, $U^2(S2, S1)$

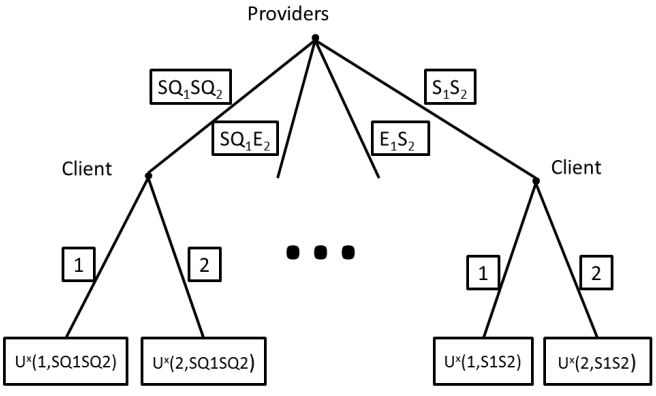


Fig. 2. The second stage of the game for two providers and one client

Based on this description, Table 2 shows an example of the provider normal-form game for two providers. The table contains the utilities for the two providers for every combination of their actions. The utilities are calculated according to Equation 3. Figure 2 shows an example of the client extensive-form game for two providers and one client. The leaf nodes contain the

utility of the client when selecting one provider given the combination of the two providers' actions. For this particular example, there $3^2 = 9$ combinations for the providers' actions.

5 Costs and Values

Having modelled the interactions among service providers and clients as a game, we develop an economic model to formalize the ecosystem in terms of values, costs, prices and development effort, capturing the effect of service evolution on these parameters. All these economic parameters of the model are defined in the context of the ecosystem considering all the externalities.

5.1 The Value of the Service

The value for the service as perceived by a client is determined by the amount of effort invested by the provider towards satisfying the client's requirements. The provider (i) estimates how much effort in person-months is required to fully satisfy global and uniform set of requirements as this was gathered from the clients and the ecosystem (M^i). The actual effort m^i that the provider invests in the evolution of its service determines the degree to which the clients' requirements are satisfied. The final value of the service is the corresponding portion of the return that the client expects from using the service (ER^x).

Any effort invested by the provider towards satisfying the clients' requirements will add value to the client. When calculating the value of a particular version of the service, the cumulative amount of effort m_o^i invested by the provider up to this version is considered. When $m^i = M^i$, then the value is equal to the client's expected return. Any additional investment by the provider will not have any effect on the value of the service as perceived by the client but they will incur additional adaptation costs, which will indirectly affect the value of the service as it will be shown in the next section. Equation 4 shows the function for the value of service.

$$V_i^x(m^i) = \begin{cases} R^x \frac{m^i}{M^i} & m^i < M^i \\ R^x & m^i \geq M^i \end{cases} \quad (4)$$

5.2 Evolution and Adaptation Costs

All software-development costs are functions of the effort in person-months and the wage w as the nominal cost for a single person-month as shown in Equation 5. Cost as a function of effort is linear.

$$C(m) = wm \quad (5)$$

The adaptation costs depend on the providers' and the clients' actions, as shown in Table 3.

5.2.1 Provider Evolves; Client Adapts

The effort required to adapt to a new version of a service is proportional to the effort required to evolve the service, since they both depend on the scope and the nature of the change itself. Further, it is assumed that it is easier to evolve the service than to adapt to it, since the provider has more information concerning the change itself. Formally, we have that $m_{ai} = am_e$, where $a > 1$. This assumption is further discussed in Section 8.

5.2.2 Provider Evolves and Supports; Client Adapts

Providers want their clients to "follow" and catch up with the evolution of their services and, in order to encourage them, they may choose to support their adaptation. In our previous work [1], we studied this process and we argued that providers are more efficient i.e., they require less effort in person-months to create adapters for their changed services than their clients, because they are more familiar with the service implementations. Since the adaptation process is specific to the service for which the adapters are created and not to the clients that use it, familiarity with the service implementation implies lower cognitive costs in the adapter creation.

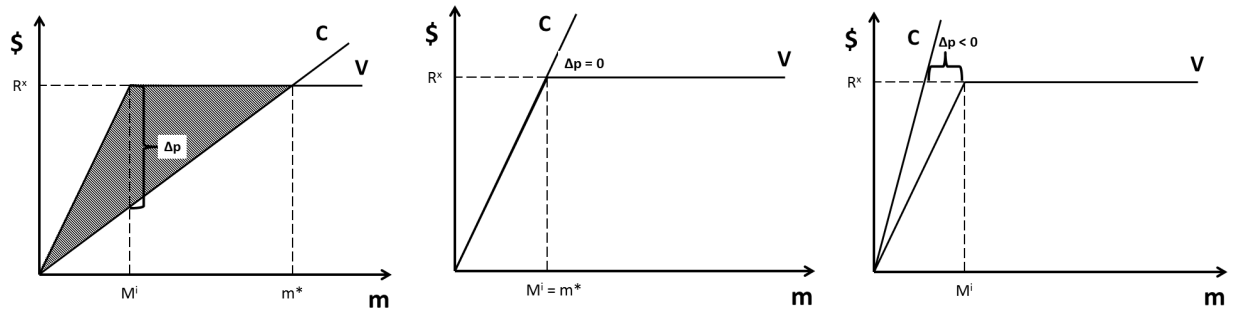
To estimate the cost of the client's effort to migrate from one version of the service to a newer one, we rely on the IBM Rational Unified Process [24] model. According to RUP, software development involves four phases (*inception*, *elaboration*, *construction* and *transition*), the second and third of which consume 75% of the total development effort. Our model assumes that the provider's inception and transition costs are negligible, since the provider has complete knowledge of the service being evolved, while, on the other hand, the client would have to assume the full costs for the adaptation. Therefore, the provider's cost of supporting the client's adaptation is 75% of the client's total adaptation cost. Eventually, what remains for the client is 25% of the original estimation for the adaptation cost.

5.2.3 Provider Evolves and Supports; Client Migrates

The third scenario involves the client abandoning its current provider, preferring an alternative service. In this case, the client's adaptation must address all the differences between the service it used to invoke and the newly chosen service. This adaptation effort is considered to be a portion of the new provider's total effort up to the version considered or, more formally, $b(m_o^j + m_e^j)$, where $b < 1$.

5.3 Cost-Value Relationship

Figure 3 illustrates the above value and cost functions. The value that the client receives from a particular service provider is maximized to ER^x when the provider's effort equals M^i. When $\frac{ER^x}{M^i} > aw^x$



(a) Feasible client. (b) Marginal client. (c) Infeasible client.

Fig. 3. The value and cost functions.

TABLE 3
Adaptation-cost functions

Provider Action	Client Adapts	Client Migrates
SQ	0	$aw^x b(m_o^j + m_e^j)$
E	$aw^x m_e^i$	$aw^x(m_e^j + b(m_o^j + m_e^j))$
S	$0.25aw^x m_e^i$	$aw^x(m_e^j + b(m_o^j + m_e^j))$

(Figure 3(a)), the cost curve intersects with that of the value when the effort equals to $m*$, as shown in the figure. If the provider's effort is less than $m*$, the client receives a value somewhere in the *efficient area* (the shaded area in the Figure), where the client is satisfied since the received value exceeds the adaptation costs. For $m < m*$, the difference between the value and the cost determines how much the price may increase for the new version while still satisfying the client. If $\frac{ER^x}{M^i} = aw^x$ (Figure 3(b)), then $m* = M^i$, which implies that this particular client cannot afford any price increase. If $\frac{ER^x}{M^i} < aw^x$ (Figure 3(c)), then the two lines do not intersect and this client can never be supported by this provider.

6 The Decision-Support System

Having discussed how software costs and values can be calculated, let us now discuss the decisions around the price of the new version of the service, the development effort and the final evolution strategy of the provider. The solution of the service-evolution game is a Nash equilibrium, where all providers have selected an evolution strategy, the clients have chosen which service to adopt, and no strategic player (provider or client) has any benefit from changing their decision.

In the context of this work, we assume that the decision around the evolution of a service is an instantaneous event. This implies that at the time of the time decision all the parameters have been considered and the environment is stable; no extra requirements are expected and no participants are expected to enter or exit the market. The implications of this assumption is further discussed in Section 8.

Furthermore, the decision on evolution is specific to a single service. If a client uses more services and the evolution of one affects the others (most probable due to quality or policy conflicts), these dependencies are factored in the adaptation costs, outside the scope of the proposed model, and are provided as input.

6.1 Price and Development Effort

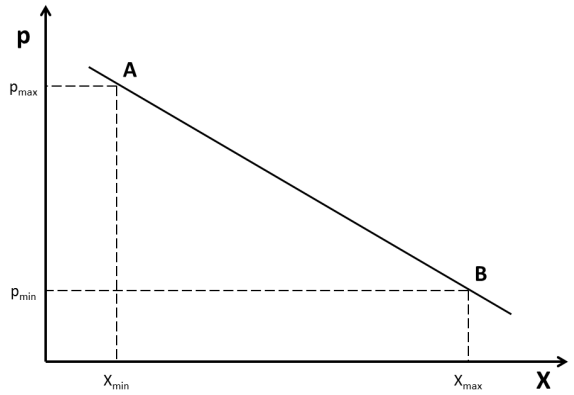
The uniformity of client requirements assumption for each service implies that all clients of the service will pay the same price for it and the provider will estimate a single value for the effort to satisfy these requirements. The optimal evolution effort is the remaining effort to fully satisfy the client requirements, $m_e^i = M^i - m_o^i$. This is the best response of a provider to the competitors' actions. If a provider invests less effort than this amount, a competitor can win over the client simply by offering a marginally better service (by slightly increasing the effort).

The service price is calculated under the assumption that there is a correlation between the price and the number of clients that use the service. If the provider increases the price of the service, the new service may be too expensive for some clients, who may choose to leave the current provider. Conversely, if the provider decreases the price, more clients may find it affordable and decide to switch to it. Therefore, the correlation between the price and the number of clients adopting the service is negative. If it is assumed that the relationship is also linear we have that:

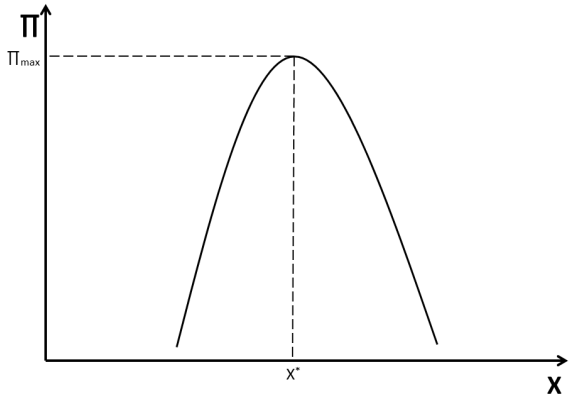
$$p^i = \alpha - \beta|X_i^t| \tag{6}$$

Although linearity is a simplification, it does not affect the steps of the analysis. Any type of relationship can be used, as long as the resulting revenue function can be maximized to find the optimal number of clients for a state.

In order to calculate the coefficients of the line, two points are needed as shown in Figure 4(a). The first point corresponds to the minimum possible price (p_{min}) to satisfy as many clients as possible (X_{max}), then the second point is the maximum possible price (p_{max}) to satisfy at least one client (X_{min}). If more clients can be satisfied by p_{max}, then $X_{min} > 1$. By solving the equation for the two points the final



(a) The price vs clients curve.



(b) The profit of the provider.

Fig. 4. The price vs clients analysis

function for the price-clients line is:

$$p^i = \frac{|X_{min}|p_{min} - |X_{max}|p_{max}}{|X_{min}| - |X_{max}|} - \frac{p_{max} - p_{min}}{|X_{min}| - |X_{max}|}|X| \quad (7)$$

From Equation 7 for price, the total revenue ($\mathcal{R}^i$) of the provider is defined as:

$$\begin{aligned}\mathcal{R}^i &= |X_i|p^i \quad (8)\\ &= \frac{|X_{min}|p_{min} - |X_{max}|p_{max}}{|X_{min}| - |X_{max}|}|X_i| - \frac{p_{max} - p_{min}}{|X_{min}| - |X_{max}|}|X_i|^2\end{aligned}$$

Next, the first derivative of Equation 8 is calculated and the optimal number of clients is the point where the revenue function is optimized. As a quadratic function, this happens when (note that the result is rounded to the closest integer to the actual result):

$$|X^*| = \left[\frac{|X_{min}|p_{min} - |X_{max}|p_{max}}{2(p_{max} - p_{min})}\right] \quad (9)$$

For each client in this set, Inequality 10 is solved for p_n^i. The left side of the inequality is the utility of the client for the provider, whose price needs to be calculated, and the right side the utility for a competitor. It is assumed that if either or both the providers evolve their service, they both try to fully satisfy the client's requirements, and so $m_e^i = M^i$. Given the competitor's price, an upper bound is calculated for the price of provider i. If there are more than one competitors, the lowest of these upper bounds is selected, because this would satisfy the client against all competitors.

$$U^x(i,s) \geq U^x(j,s) \quad (10)$$

If the X^* set is sorted according to the upper bounds calculated for each client in descending order, then the optimal price is that of the last client in the sorted set. This method is applied for all states of the game so that an optimal price is calculated according to all the competitors' different actions.

6.2 The Nash Equilibrium

Having calculated the right amount of effort to invest in the service evolution and the right price to ask for the evolved version, the second stage of the game (Figure 2) is solved to determine the new client distribution among the providers. For each state of the provider game, the utility of a client is calculated for each provider and they are compared with each other. The maximum utility determines the provider whose service the client will use for the particular state. The aggregate results of the second stage, for all clients and states of the game, determines each provider's clients for each state.

Note that the price has been calculated *ceteris paribus*, i.e., considering the estimated future price for the competitor as a given. If the competitor's future price was to be considered as a variable and subject to calculation, its calculation would create an infinite loop; as a result, one should note that the actual set of the provider's clients may differ from the set determined with X^*.

With the actual price, the actual evolution effort and the actual set of clients for each provider for each state of the game, a complete picture of the state of the game i.e., the ecosystem after the service evolution is formed. The extra incentive behind modelling service evolution as a game is that, as a finite game, it is guaranteed to have at least one solution in the form of a Nash equilibrium as it has been proven [25]. The resulting game is a N-player general-sum game (depending on the number of providers). Such games are PPAD-complete [26] and cannot be solved in polynomial time. However, if the number of providers is relatively small, the following steps can be taken to compute an equilibrium.

6.2.1 Dominated-Strategies Removal

The first step involves the iterative removal of dominated strategies of the providers. The first provider's actions are examined and if a dominated one is found,

it is removed from this provider's action set. Given the updated action set, the second provider's actions are examined in a similar way. After all providers have been examined, the process is repeated starting again from the first provider. The process is repeated until only one action remains for each provider or no actions are removed in a cycle (i.e., a single iteration of all the providers).

6.2.2 Mixed Nash Equilibrium

If the above process has not resulted in a single action for each provider, the next step attempts to identify a mixed-strategy Nash equilibrium, where the providers randomly decide on an action according to a given probability distribution. To find the distribution for a given provider, probability variables are assigned to the actions, e.g. $s_i = \{p, q, 1-p-q\}$[1]. The distribution that corresponds to an equilibrium is that for which the other providers become indifferent between their actions, formally, $U^j(SQ_j, s_{-j}) = U^j(E_j, s_{-j}) = U^j(S_j, s_{-j})$. This system is solved to find the probability distribution. If no proper probability distribution is found, this implies that there might exist dominated strategies. In order to find the potential dominated strategies, a subsystem is solved with only two of the three actions of provider j's actions. If the probability of one of the actions is found to be negative in all subsystems that it participates, then this action is dominated and it can be removed. The process repeats until a mixed strategy equilibrium is found or no more actions can be removed.

6.2.3 Weak Nash Equilibrium

If an equilibrium cannot be found with either of the above methods, an extraneous condition may be imposed. The action combinations that satisfy the condition may not correspond to a true equilibrium, but they still represent valid decisions.

One such condition is that the providers form a cartel. In a cartel, the providers will agree on specific evolution strategies that maximize the total payoff for the collective of the providers. This may be a fragile collaboration, since there can exist at least one provider who will be benefited by switching to a different strategy.

An alternative extraneous condition is to maximize the total welfare of the ecosystem. The welfare of the ecosystem is defined as the sum of all the providers' and clients' utilities. Such an equilibrium may create opportunities for collaboration between providers and clients. This can be a realistic scenario for small ecosystems or for very tightly dependent software components that may dictate stronger relationships and collaboration between the providers and the clients.

1. Each probability is a positive number less than 1 and all together add up to 1.

7 The Cloud-Services Case Study

We illustrate our decision-support system with a synthetic but realistic example, inspired by the ecosystem of PaaS (Platform-as-a-Service) and IaaS (Infrastructure-as-a-Service) cloud providers. We have three fictional providers: Amzn, Macroshift and Goggle. In this ecosystem, we consider a synthetic population of five clients with the same set of requirements as per the uniformity assumption.

Even though the example case study is synthetic, it serves to illustrate two important points regarding our contribution. First, the case study helps us walk through the steps of our decision-support method and discuss the implications of its suggestions. Second, and perhaps more importantly, it illustrates that the values assumed as input by our decision-support methodology are available, some in publicly accessible resources (like the developers' wages) and some are part of the participating organization's record (like the effort invested in developing the current version of the service) (see also discussion in Section 8).

7.1 The Ecosystem Context

Each provider offers software services to manage virtual machines and images on their respective cloud infrastructures. At the point where evolution is considered, the three providers offer slightly different features and, therefore, each provider requires different amounts of effort to fully satisfy the clients' requirements. Table 4 shows the effort that each of these providers has invested in their offerings up to now, the effort required to fully satisfy the clients' requirements, and the effort required to create adapters for these clients once the services have evolved. The values for the current and total effort were randomly generated from normal distributions (between 5 to 12 person-months and 15 to 20 person-months respectively). The adaptation effort is calculated based on the service-evolution effort, assuming a factor $a = 1.2$ ($m_{ai} = am_e^i$).

Table 5 shows prices, wages and the current market division for the three providers in our example. The prices that the providers charge their clients for the purchase of virtual-machine instances were obtained from the web sites of three actual service providers and correspond to five medium to high capability instances. The wages were obtained from Glassdoor[2], a web site that contains information (including salaries) from employers worldwide. For this example, we use wages that correspond to a medium-level software engineer. The market is roughly divided based on a recent market surveys [27], according to which three major providers have 53%, 12% and 10% of the market share.

For the clients of the ecosystem, we have randomly generated their expected return to be around \$250K

2. http://www.glassdoor.com/index.htm

TABLE 4
Current, total and adaptation effort in person-months for the cloud-services ecosystem

Providers	Current (m_o^i)	Total (M^i)	Adaptation (m_{ai}^i)
Amzn	10	16	7
Macroshift	7	19	14
Goggle	10	18	9

TABLE 5
Providers: Prices, wages, and market division

Providers	Current Price (p_o^i)	Future Price (p_n^i)	Wage (w^i)	Market division (X^i)
Amzn	$31,859.60	$147,312.00	$9,925.00	A,B,C
Macroshift	$30,127.40	$145,843.20	$8,658.10	D
Goggle	$36,310.20	$84,283.20	$9,9 16.33	E

and the wages of their software developers to be between $5,000 and $10,000 as shown in Table 6.

Table 7 shows the estimated adaptation effort for each client-provider combination, according to the market division, the provider's evolution and adaptation effort based on the functions from Table 3.

7.2 Solving the Example

Having established the market division of the service ecosystem, the amount of effort invested in developing the various service offerings and their current prices, the wages of the software developers, and the amounts of effort necessary to fully meet the anticipated client requirements, we can now proceed to solve the service-evolution game.

First, we find the optimal price and the evolution effort of the providers for each state of the game. We will demonstrate the process for a single game state as an example. In this example state, Amzn chooses to evolve (E), Macroshift chooses to support its clients (S) and Goggle chooses to retain the status quo of the service (SQ). Since the provider game is simultaneous and the providers decide independently and in the same way for each state of the game, the same process is applied for all states of the game.

An additional piece of information needed for this analysis is the price for each service after they have evolved. Although this price is the result of the analysis, when considering it from a single provider's point of view it is required that the competitors' prices be known. For this reason, estimates can be used that can be the results of regression analysis of historic data for example. In our case study, the instances that will come from the evolution process of the three providers already exist in reality. Therefore, we can use those prices from the providers' web sites as the future prices that are shown in Table 5. Again, the table contains the total yearly cost for 5 instances.

As an example, we will analyse the situation from Amzn's perspective as the decision maker, to find the optimal price for this state for Amzn. We follow the same process for all other providers. Table 8 shows the upper bounds calculated for the Amzn price for all competitor-client pairs.

TABLE 6
Clients: Expected returns and wages

Clients	Expected Return (ER^x)	Wage (w^x)
A	$254,903.80	$8,013.10
B	$268,499.60	$7,629.72
C	$289,261.40	$7,337.00
D	$257,957.50	$6,604.00
E	$258,585.30	$5,991.41

TABLE 7
Client-adaptation effort

	Amzn			Macroshift			Goggle		
Clients	SQ	E	S	SQ	E	S	SQ	E	S
A	0	8	2	4	18	18	5	15	15
B	0	8	2	4	18	18	5	15	15
C	0	8	2	4	18	18	5	15	15
D	5	13	13	0	15	4	4	15	15
E	5	13	13	4	18	18	0	10	3

TABLE 8
Price bounds for Amzn

Clients	Macroshift	Goggle
A	$244,901.17	$102,090.45
B	$246,779.37	$100,734.33
C	$250,876.98	$100,419.80
D	$172,405.66	$54,223.92
E	$201,406.20	$18,319.05

According to Table 8 and Equation 9, the optimal number of clients in this case is 3, with optimal price at $\$100,419.80$ for Amzn. Similarly, the optimal number of clients and the optimal price for Macroshift are 1 and $\$27,661.45$ respectively. We don't need to calculate a new price for Goggle, because in this particular state the Goggle service doesn't change. We use these prices in the second stage of the game. The result of this game will give the new market division for this state which is: Amzn={A,B,C}, Macroshift={D,E} and Goggle={}.

At first glance, it appears that retaining the status quo in an evolving environment proved to be a mistake for Goggle, while choosing to support its clients adaptation allowed Macroshift to retain its client against the stronger competitor, Amzn. Macroshift also managed to claim client E from Goggle over Amzn, in spite of the higher adaptation costs, since Macroshift could offer the same service in a more competitive price for client E than Amzn. This did not hurt Amzn, because they managed to retain their original clients for a higher price. They would not be in a better position by lowering their price just to satisfy client E. This example shows that our approach produces prices that result in a balanced ecosystem for

TABLE 9
The provider game for cloud services

	SQ2	E2	S2
SQ3			
SQ1	$95,578.80	$95,578.80	$95,578.80
E1	$95,578.80	$95,578.80	$95,578.80
S1	$95,578.80	$95,578.80	$95,578.80
E3			
SQ1	$342,129.19	$139,326.16	$139,326.16
E1	$241,709.39	-$59,550.00	-$59,550.00
S1	$241,709.39	-$59,550.00	-$59,550.00
S3			
SQ1	$270,669.19	$67,866.16	$67,866.16
E1	$170,249.39	$175,580.75	$175,580.75
S1	$170,249.39	$175,580.75	$175,580.75

	SQ2	E2	S2
SQ3			
SQ1	$30,127.40	$0.00	$0.00
E1	-$93,302.93	-$98,600.06	-$98,600.06
S1	-$217,979.57	-$223,276.70	-$223,276.70
E3			
SQ1	$0.00	$0.00	$0.00
E1	-$48,574.30	-$60,981.01	-$60,981.01
S1	-$173,250.94	-$185,657.65	-$185,657.65
S3			
SQ1	$0.00	$0.00	$0.00
E1	-$48,574.30	-$32,332.93	-$32,332.93
S1	-$173,250.94	-$157,009.57	-$157,009.57

	SQ2	E2	S2
SQ3			
SQ1	$36,310.20	$20,370.85	-$74,825.91
E1	$0.00	-$29,479.89	-$124,676.66
S1	$0.00	-$29,479.89	-$124,676.66
E3			
SQ1	$36,310.20	-$1,778.34	-$96,975.11
E1	$0.00	-$35,693.50	-$130,890.27
S1	$0.00	-$42,196.13	-$137,392.90
S3			
SQ1	$36,310.20	-$1,778.34	-$96,975.11
E1	$0.00	$882.38	-$94,314.39
S1	$0.00	-$63,984.82	-$159,181.59

the particular state of the game.

Having the prices and the clients determined for all providers and all game states we can now calculate the provider utilities for the first stage of the game. Table 9[3] shows the normal form game for the providers of the example with all the utilities fully calculated. The first set of three tables contain the utilities for Amzn, the second for Macroshift and the last one for Goggle. In each table of each set the provider actions are indexed according to which provider takes what action; 1 is for Amzn, 2 is Macroshift and 3 is for Goggle.

Table 9 shows that there are no dominated strategies for any of the providers. However, looking at the total payoffs of all the providers for each state in Table 10, it appears that there is a state $(SQ1, SQ2, E3)$, that maximizes the providers' total payoff. Incidentally, this point is also a Nash equilibrium as no provider would gain anything from switching to another action. At

3. Table 9 is an instance of Table 2.

TABLE 10
Total provider payoffs for the provider game

	SQ2	E2	S2
SQ3			
SQ1	$162,016.40	$115,949.66	$20,752.89
E1	$2,275.87	-$32,501.16	-$127,697.92
S1	-$122,400.77	-$157,177.80	-$252,374.56
E3			
SQ1	$378,439.39	$137,547.81	$42,351.05
E1	$193,135.09	-$156,224.51	-$251,421.28
S1	$68,458.45	-$287,403.79	-$382,600.55
S3			
SQ1	$306,979.39	$66,087.81	-$29,108.95
E1	$121,675.09	$144,130.20	$48,933.43
S1	-$3,001.55	-$45,413.64	-$140,610.41

TABLE 11
Total payoffs for the ecosystem

	SQ2	E2	S2
SQ3			
SQ1	$746,610.97	$744,412.95	$692,354.33
E1	$679,523.64	$688,615.32	$636,556.71
S1	$626,170.20	$635,261.88	$583,203.27
E3			
SQ1	$896,647.97	$813,082.94	$761,024.33
E1	$781,632.55	$493,956.31	$441,897.70
S1	$728,279.11	$440,602.87	$388,544.26
S3			
SQ1	$949,278.99	$865,713.97	$813,655.35
E1	$834,263.58	$820,990.95	$768,932.33
S1	$780,910.14	$767,637.51	$715,578.89

the very least Macroshift and Goggle are indifferent to their actions while Amzn retains its status quo.

Examining the total payoffs for all the participants of the ecosystem (providers and clients) in Table 11, one can see that the payoff is maximized in the state $SQ1, SQ2, S3$. Again, this is another Nash equilibrium for the same reason as before. This state results in a smaller payoff for the providers than the previous equilibrium but more for the clients. This can be an opportunity for collaboration among the providers, who can decide to simultaneously increase their prices until the market division remains the same. This would result in a cooperative game, which is a topic for future research and it is not covered in this work.

8 Assumptions and Threats to Validity

The decision-support system that we have described in this paper examines a service ecosystem circumscribed by three basic assumptions.

Uniformity of client requirements. This assumption is logically based on how a provider may gather and implement client requirements. In practice, a provider will not create one service for each client, but rather a service with multiple features that can satisfy a larger number of clients. If the requirements start to differ significantly, then the provider may decide to split the offerings into two services, which will then be

charged differently. For example, in our case study, Amzn may offer on-demand instances charged on a per-hour basis, reserved instances for a fixed price or spot instances charged based on an auction-scheme[4]. Therefore, since we are talking about different services and different prices, the evolution of each one of them will be addressed as a different game with different clients for each one of them. If changes in the services affect each other, then the provider can play the client game for each service and accumulate its outcomes on the provider game. From a client's perspective, the difference in the perception of the features' implementation by the providers is included in the expected return. In the future, we plan to investigate how the expected return is estimated by each functional and non-functional requirement of a client.

Paid services. This business relationship between providers and clients creates contractual dependencies between the participants, thus forcing the providers to consider the externalities of their decisions concerning the evolution of their services. The strong relationships in the examined ecosystem also imply that providers and clients are close enough to exchange information that may be important for service-evolution decisions. This information includes values for the input variables of the decision support system. Although we recognize the existence of other pricing mechanisms, such as the freemium model and the advertisement-based model, these ecosystems imply different, more subtle, externalities than what we considered in this paper.

Instantaneous nature of service evolution. The basis of this assumption is that the volatility of the service ecosystem is very hard to predict and therefore any decision would be based on very rough estimates. Therefore, it is preferable to constrain the knowledge about the ecosystem on a short period and make decisions based on whatever is known or it can easily be estimated. The implication of this assumption is that any long-term effects are not factored in the decision process and this is something we plan to explore in the future but rather as a complement to the constrained decision due to the sensitivity of the associated predictions.

Another threat to validity is implied by the assumptions concerning the the knowledge/availability of certain economic parameters in our models.

ER^x. The client's expected return for a service is considered to be the opportunity cost of choosing to pay for the service, as opposed to developing an in-house solution. If the in-house solution is the more expensive option, then the anticipated return corresponds to the funds the client saves. For *target* clients (i.e., clients that the provider tries to "steal" from competitors), these values could be estimated by taking advantage of publicly available information, such as marketing surveys or SLAs (Service Level Agreements), which can indicate how much clients value similar services from the domain and also how different clients value each feature offered by the services from the ecosystem.

M^i. The client communicates its requirements to the provider by making direct feature requests or, conversely, the provider solicits these requirements from market research. The provider then estimates how much effort is required to implement the new features and this effort is added to whatever effort was already invested up to the current version of the service to comprise the M^i variable.

p^i and w. Software prices and wages for software developers are usually public information, easy to obtain or, at least, estimated. In order to estimate future price of competitive services (i.e., after evolution), the providers can perform regression analysis on utilize historic data. Future value of a competitive service can be assessed by studying the new features and improvements in quality properties of the service.

m_e^j. The competitors development effort can be estimates by taking into account information from public software artifacts including service interfaces and possibly business processes. In our previous work [28], we have initiated a discussion about a possible way of assessing the impact of changes in the service interface to client applications. This assessment can be quantified into a metric, which enables us to estimate the cost of a change, based on the abstract information conveyed by the service interface. Assuming that services offer similar features, a provider may estimate the effort required by his competitors to implement those features. The differences in processes or development capabilities can be inferred from public sources and used in the estimation.

m_a^x. The idea that adaptation effort depends on the changes of the service (i.e., the evolution effort) is fairly broadly accepted. In fact, it is the key intuition for estimating testing effort, based on software changes using, for example, *Function Point Analysis* (FPA) [29]. Similar techniques can be used to estimate adaptation effort from changes in the service, its size and its new functionality.

In general, the manner in which the input variables are calculated does not affect the robustness of the models, which are valid by construction. Clearly, the actual values of the input variables will affect the final decision, which is the objective of the proposed models after all.

9 Conclusion

In this work, we emphasize the need to study service evolution taking into account the broader context of the business ecosystem implied by the software interdependencies between service providers and service clients. The evolution of services may exhibit

4. http://aws.amazon.com/ec2/purchasing-options/

non-typical, and even exaggerated, consequences to this ecosystem. For example, a small change on the service side may result in a series of complicated and expensive changes on the client side. Continuous incompatible changes to the service may cause the client to distrust the provider. The potential loss of income from clients switching providers is not currently accounted for when a provider considers the evolution of a service. Thus, service evolution requires novel support methods, to balance the interests of all the participants involved, and to increase the value they receive from the innovations brought about by the evolution.

The work presented in this paper makes three contributions towards this objective. We first developed a game-theoretic model to capture the relationships and complex interactions between the self-interested partners of the service ecosystem, which includes competing providers and clients. Second, we formulated an economics model to calculate the relevant economic parameters when considering the evolution of a service. Finally, we presented a decision-support method to assist providers in estimating the optimal levels of the effort they need to invest in evolving their services and supporting the adaptation of their clients, as well as, the optimal price they can demand for their new service versions. We have demonstrated this methodology with a synthetic yet realistic case study.

Our framework substantially informs the decision-making process of service providers, who, originally, might have only considered their services, the development costs associated with the evolution, and potential value of the evolution in terms of additional income to be accrued due to increased prices to be commanded for the additional service features. Such a simpler problem formulation, ignorant of the externalities implied by the business relationships between providers and clients, is bound to lead to "naive" decisions with potentially negative results for the ecosystem. With our deeper analysis of the decision-making context, the provider can reach a decision that maximizes its own revenue, while, at the same time, maintaining the stability in the ecosystem as a whole.

REFERENCES

[1] M. Fokaefs, E. Stroulia, and P. R. Messinger, "Software Evolution in the Presence of Externalities: A Game-Theoretic Approach," in *Economics-Driven Software Architecture*, I. Mistrik, R. Bahsoon, R. Kazman, K. Sullivan, and Y. Zhang, Eds. Elsevier, 2013.

[2] J. Laffont, "externalities," in *The New Palgrave Dictionary of Economics*, S. N. Durlauf and L. E. Blume, Eds. Basingstoke: Palgrave Macmillan, 2008.

[3] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice Hall, 1981.

[4] B. Boehm and K. Sullivan, "Software economics: status and prospects," *Information and Software Technology*, vol. 41, no. 14, pp. 937–946, 1999.

[5] B. W. Boehm and K. J. Sullivan, "Software economics: A roadmap," in *Proceedings of the Conference on the Future of Software Engineering*. ACM Press, 2000, pp. 319–343.

[6] B. Tansey, *Valuing software services: The real options-based modularity analysis framework*. University of Alberta (Canada), 2008.

[7] M. Lanza, R. Marinescu, and S. Ducasse, *Object-oriented metrics in practice*. Springer, 2006.

[8] I. Ozkaya, R. Kazman, and M. Klein, "Quality-Attribute Based Economic Valuation of Architectural Patterns," in *2007 First International Workshop on the Economics of Software and Computation*. IEEE, May 2007, pp. 5–5.

[9] B. W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, R. Madachy, and B. Steece, *Software Cost Estimation with Cocomo II with Cdrom*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[10] K. Lyons, C. Playford, P. Messinger, R. Niu, and E. Stroulia, "Business models in emerging online services," in *Value Creation in E-Business Management*, ser. Lecture Notes in Business Information Processing, M. Nelson, M. Shaw, and T. Strader, Eds. Springer Berlin Heidelberg, 2009, vol. 36, pp. 44–55.

[11] K. Lyons, P. R. Messinger, R. H. Niu, and E. Stroulia, "A tale of two pricing systems for services," *Inf. Syst. E-bus. Manag.*, vol. 10, no. 1, pp. 19–42, Mar. 2012.

[12] R. H. Coase, "The Problem of Social Cost," *Journal of Law and Economics*, vol. 3, pp. 1–44, 1960.

[13] W. H. Hoffmann, "Strategies for managing a portfolio of alliances," *Strategic Management Journal*, vol. 28, no. 8, pp. 827–856, Aug. 2007.

[14] A. Barros, M. Dumas, and P. Bruza, "The move to Web service ecosystems," *BPTrends*, no. September, pp. 1–9, 2005.

[15] A. Barros and M. Dumas, "The Rise of Web Service Ecosystems," *IT Professional*, vol. 8, no. 5, pp. 31–37, Sep. 2006.

[16] N. S. Caswell, C. Nikolaou, J. Sairamesh, M. Bitsaki, G. D. Koutras, and G. Iacovidis, "Estimating value in service systems: A case study of a repair service system," *IBM Systems Journal*, vol. 47, no. 1, pp. 87–100, 2008.

[17] K. Chow and D. Notkin, "Semi-automatic update of applications in response to library changes," in *Software Maintenance 1996, Proceedings., International Conference on*. IEEE, 1996, pp. 359–368.

[18] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani, "Developing adapters for web services integration," in *CAiSE*, 2005, pp. 415–429.

[19] P. Kaminski, M. Litoiu, and H. Müller, "A design technique for evolving web services," in *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research - CASCON '06*. New York, New York, USA: ACM Press, Oct. 2006, p. 23.

[20] S. R. Ponnekanti and A. Fox, "Interoperability among independently evolving web services," in *Middleware '04*. New York, NY, USA: Springer-Verlag New York, Inc., 2004, pp. 331–351.

[21] L. Maciaszek, *Requirements analysis and system design*. Pearson Education, 2007.

[22] G. Karner, "Resource estimation for objectory projects," *Objective Systems SF AB*, vol. 17, 1993.

[23] A. Albrecht, "Measuring application development productivity," in *Proceedings of IBM Applic. Dev. Joint SHARE/GUIDE Symposium*, Monterey, CA, USA, 1979, pp. 83–92.

[24] P. Kruchten, *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[25] J. F. Nash *et al.*, "Equilibrium points in n-person games," *Proceedings of the national academy of sciences*, vol. 36, no. 1, pp. 48–49, 1950.

[26] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, "The complexity of computing a nash equilibrium," *SIAM Journal on Computing*, vol. 39, no. 1, pp. 195–259, 2009.

[27] J. Bort, "Amazon is crushing ibm, microsoft, and google in cloud computing, says report," web site, November 2013, last accessed 2014-04-13.

[28] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau, "An Empirical Study on Web Service Evolution," in *Proceedings of the 2011 IEEE International Conference on Web Services*, ser. ICWS '11, Washington, DC, USA, 2011, pp. 49–56.

[29] C. R. Symons, "Function point analysis: difficulties and improvements," *Software Engineering, IEEE Transactions on*, vol. 14, no. 1, pp. 2–11, 1988.

Chapter 5

Conclusions and Future Work

5.1 Summary and Conclusions

Software evolution presents special challenges in the context of service-oriented systems. This is not only because of their modular and distributed nature, but also due to the very nature of the interaction between providers and clients: clients only have access to the provided service-interface specification (the service implementation is hidden) and need a "live" run-time connection with the service (they cannot own "local" service copies). Given these constraints, best practices, like backwards compatibility, are not or may not always be followed. Recognizing this phenomenon, in my thesis, I have developed WSDarwin, a set of automated and interactive tools to support both providers and clients during the evolution of service-oriented systems.

My work makes three contributions towards addressing the problem of service-oriented system evolution.

1. I have developed tools and methods to support the adaptation of client applications in response to evolved WS-* services (RP2-WS) (Chapter 2). The WS-* toolkit allows developers to identify changes in a web service by comparing different versions of its WSDL interface (Section 2.3) based on WSMeta, a compact and concise service interface meta-model (Section 2.2). The results of the comparison are then used to adapt client applications to the new service version (Section 2.4). The toolkit is implemented as an Eclipse plug-in so that its offerings are inte-

grated with the development environment and allow for testing the client application after the adaptation process (Section 2.5). The automation and integration of the plug-in aims at minimizing the developers' effort to ensure backwards compatibility for client applications between two versions of a web service.

2. I have developed tools and methods to support the development and maintenance of REST client applications (RP1-REST) (Chapter 3). The REST toolkit enables the automatic generation of WADL interfaces for REST services based on example invocations, which are typically available in the REST-service developer documentation. Based on the generated WADL interface specification, the automatic generation of a Java client proxy becomes possible to mediate the client's use of the service (Section 3.1). The toolkit also offers version-comparison capabilities and cross-vendor service mapping to facilitate the migration of client applications between similar services offered by different providers (Section 3.2). The toolkit is implemented in a web application offering a simple and easy-to-use interface (Section 3.3). The simplicity of the web application provides automation for the development and maintenance of REST applications without adding overhead to developers.

3. I have developed a system to support the service-evolution decision-making process taking into account both the technical and economic implications of evolution (RP2) (Chapter 4). This is achieved through an economic model to calculate all the economic parameters of the evolution and a game-theoretic model to capture the technical and business interactions between providers and clients (Sections 4.1 and 4.3). The models enable decision-support systems for service ecosystems of various complexities (Sections 4.2 and 4.3). The systems can lead the ecosystem participants to outcomes that are efficient with respect to technical, social and financial criteria.

The thesis of my work is that service evolution can and should be not only technically but also socially and economically conscious through support

from automated tools. WSDarwin demonstrates the plausibility of this statement. The anticipated impact of my work is to increase, through tool support, the systematicity of service evolution and client adaptation. The automation will minimize the effort required to ensure backwards compatibility between different versions of a web service. Tool support implies that developers of client applications will now be able to efficiently and effectively react to service changes even if evolution standards are not followed completely. With respect to the decision-making process, my work will enable stakeholders to better budget the evolution of service-oriented systems by taking into account not only the direct and technical implications of their decisions but also the indirect and economical ones. This will also help the participants of a web service ecosystem to form stronger business relationships with their partners.

5.2 Future Plans and Directions

Through my work, one of my goals is to emphasize on the importance of structured software interfaces and automatically generated code. The uniformity that these artifacts provide can significantly facilitate automation for many development tasks. Service evolution is one of those tasks and I have demonstrated in this dissertation how it can be systematized through support from automated and interactive tools. Other tasks such as service discovery, selection and migration can be further automated. Interfaces are between the few artifacts that are publicly available within a service ecosystem and the way they are consumed by auto-generated client proxies is also common knowledge. I am interested in using this public knowledge to enable developers estimate the cost of software development and evolution without having extended knowledge about the specifics of software components that might not be publicly available. This estimation can also help the decision-making process as it was discussed in Chapter 4.

All of the methods proposed in this dissertation or discussed as future work can, in principal, be applied on any software system or development paradigms that rely on structured interfaces and auto-generated source code. Model-

driven software development is such a paradigm. The ability to track and propagate changes along the various layers of models and meta-models is highly desirable in model-driven techniques. Tools that identify differences in abstract and standard interfaces, like those of models, and adapt the auto-generated instances to these changes could automate this functionality. Other fields where the methods and tools of this thesis can be useful include mobile and cloud applications. Since such systems rely heavily on standard and structure interfaces for their development, deployment and configuration, tools like the comparator and the mapper can be used to systematically migrate applications between platforms.

Another goal of my work is to raise awareness about the economic and social implications of software development and maintenance. This particular question became rather popular during the 80's when software started to be developed on an industrial scale and became of critical importance for many tasks. However, ever since the research on software economics reached a plateau, despite the fact that new technologies, new processes and new software architectures have emerged. These novelties have made most of the previous methods and considerations to become outdated. The degree of distribution of modern architectures and the variability of their components imply that decisions on economic issues have to now be made under greater uncertainty with minimal information. The ideas proposed in this work, for example using Game Theory as a tool to capture the interactions between these separate and independent entities within the software ecosystem and consider the externalities of their decisions, can address many of the issues introduced by modern architectures. In combination with other traditional software economics techniques, including cost estimation and software pricing models, the methods I propose can be applied on a variety of modern software engineering problems. For example, in the domain of cloud computing, problems like cloud provider selection and migration, the vendor lock-in problem and pricing of virtual infrastructures are very similar in nature as the ones discussed in this dissertation. Similarly, in the domain of mobile computing, there are problems like mobile platform selection and migration and developing native applications

against generic web applications are problems with economic consideration that can benefit from methods such as the ones proposed here.

Bibliography

Albrecht, A., 1979. Measuring application development productivity. In: Proceedings of IBM Applic. Dev. Joint SHARE/GUIDE Symposium. Monterey, CA, USA, pp. 83–92.

Ali, N., Babar, M. A., 2009. Modeling service oriented architectures of mobile applications by extending soaml with ambients. In: Euromicro Conference on Software Engineering and Advanced Applications. SEAA '09. pp. 442–449.

Ali, N., Nellipaiappan, R., Chandran, R., Babar, M. A., 2010. Model driven support for the service oriented architecture modeling language. In: International Workshop on Principles of Engineering Service-Oriented Systems. PESOS '10. pp. 8–14.

Alonso, G., Casati, F., Kuno, H., Machiraju, V., 2004. Web services: concepts, architectures and applications. Springer.

Anderson, T., April 2006. Ws-* vs the rest. `http://www.theregister.co.uk/2006/04/29/oreilly_amazon/`.

Andrikopoulos, V., Benbernou, S., Papazoglou, M. P., 2008. Managing the evolution of service specifications. In: International Conference on Advanced Information Systems Engineering (CAiSE 2008). Springer-Verlag, Berlin, Heidelberg, pp. 359–374.

Apache, S. F., 2012. Apache Axis2/Java. `http://axis.apache.org/axis2/java/core/`.

Aversano, L., Bruno, M., Canfora, G., Di Penta, M., Distante, D., 2006. Using Concept Lattices to Support Service Selection. International Journal of Web Services Research 3 (4), 32–51.

Aversano, L., Bruno, M., Penta, M. D., Falanga, A., Scognamiglio, R., 2005. Visualizing the Evolution of Web Services using Formal Concept Analysis. International Workshop on Principles of Software Evolution, 57–60.

Barros, A., Dumas, M., Sep. 2006. The Rise of Web Service Ecosystems. IT Professional 8 (5), 31–37.

Barros, A., Dumas, M., Bruza, P., September 2005. The move to Web service ecosystems. BPTrends 3 (3), 1–9.

Bazelli, B., Fokaefs, M., Stroulia, E., 2013. Mapping the responses of restful services based on their values. In: IEEE International Symposium on Web Systems Evolution (WSE 2013). IEEE, pp. 15–24.

Benatallah, B., Casati, F., Grigori, D., Nezhad, H. R. M., Toumani, F., 2005. Developing adapters for web services integration. In: International Conference on Advanced Information Systems Engineering (CAiSE 2014). pp. 415–429.

Boehm, B., Sullivan, K., 1999. Software economics: status and prospects. Information and Software Technology 41 (14), 937–946.

Boehm, B. W., 1981. Software Engineering Economics. Prentice Hall, Englewood Cliffs, NJ.

Boehm, B. W., Clark, Horowitz, Brown, Reifer, Chulani, Madachy, R., Steece, B., 2000. Software Cost Estimation with Cocomo II with Cdrom, 1st Edition. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Boehm, B. W., Sullivan, K. J., 2000. Software economics: A roadmap. In: Proceedings of the Conference on the Future of Software Engineering. ACM Press, pp. 319–343.

Bordbar, B., Staikopoulos, A., September 2004. Automated generation of metamodels forweb service languages. In: European Workshop on Model Driven Architecture. MDA'04.

Bort, J., November 2013. Amazon is crushing ibm, microsoft, and google in cloud computing, says report. web site, last accessed 2014-04-13.

Boyd, S., Vandenberghe, L., 2004. Convex Optimization. Cambridge University Press, New York, NY, USA.

Cao, F., Bryant, B. R., Zhao, W., Burt, C. C., Raje, R. R., Olson, A. M., Auguston, M., 2004. A meta-modeling approach to web services. In: IEEE International Conference on Web Services (ICWS 2004). ICWS '04. pp. 796–800.

Caswell, N. S., Nikolaou, C., Sairamesh, J., Bitsaki, M., Koutras, G. D., Iacovidis, G., 2008. Estimating value in service systems: A case study of a repair service system. IBM Systems Journal 47 (1), 87–100.

Chawathe, S. S., Rajaraman, A., Garcia-Molina, H., Widom, J., 1996. Change Detection in Hierarchically Structured Information. ACM Sigmod Internation Conference on Management of Data, 493–504.

Chemuturi, M., 2009. Software estimation best practices, tools & techniques: a complete guide for software project estimators. J. Ross Publishing.

Choi, S. C., 1991. Price Competition in a Channel Structure with a Common Retailer. Marketing Science 10 (4), 271–296.

Chow, K., Notkin, D., 1996. Semi-automatic update of applications in response to library changes. In: Internation Conference on Software Maintenance (ICSM 1996). IEEE, pp. 359–368.

Coase, R. H., 1960. The Problem of Social Cost. Journal of Law and Economics 3, 1–44.

CXF, A., 2013. wadl2java command line tool. `http://cxf.apache.org/docs/jaxrs-services-description.html#JAXRSServicesDescription-wadl2javacommandlinetool`.

Daskalakis, C., Goldberg, P. W., Papadimitriou, C. H., 2009. The complexity of computing a nash equilibrium. SIAM Journal on Computing 39 (1), 195–259.

Developers, G., October 2014. The Google Geocoding API. `https://developers.google.com/maps/documentation/geocoding/`.

Dulucq, S., Tichit, L., September 2003. Rna secondary structure comparison: exact analysis of the zhang–shasha tree edit algorithm. Theoretical Computer Science 306, 471–484.

Elio, R., Stroulia, E., Blanchet, W., Apr. 2009. Using interaction models to detect and resolve inconsistencies in evolving service compositions. Web Intellgence and Agent Systems 7 (2), 139–160.

Espinha, T., Zaidman, A., Gross, H.-G., 2014. Web api growing pains: Loosely coupled yet strongly tied. Journal of Systems and Software.

Fensel, D., Bussler, C., 2002. The web service modeling framework wsmf. Electronic Commerce Research and Applications 1 (2), 113–137.

Fielding, R. T., 2000. REST: architectural styles and the design of network-based software architectures. Doctoral dissertation, University of California, Irvine.

Fluri, B., Würsch, M., Pinzger, M., Gall, H. C., 2007. Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction. IEEE Transactions on Software Engineering 33 (11), 725–743.

Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A., 2011. An Empirical Study on Web Service Evolution. In: IEEE International Conference on Web Services (ICWS 2011). ICWS '11. Washington, DC, USA, pp. 49–56.

Fokaefs, M., Stroulia, E., 2012. WSDarwin: automatic web service client adaptation. In: Conference of Center for Advanced Studies (CASCON 2012). pp. 176–191.

Fokaefs, M., Stroulia, E., 2013a. Wsdarwin: A decision-support tool for web-service evolution. In: IEEE International Conference on Software Maintenance, Early Research Achievement (ICSM 2013 ERA). IEEE, pp. 444–447.

Fokaefs, M., Stroulia, E., 2013b. WSMeta: a meta-model for web services to compare service interfaces. In: Panhellenic Conference on Informatics (PCI 2013). ACM, pp. 1–8.

Fokaefs, M., Stroulia, E., June 2014a. The WSDarwin Toolkit for Service-Client Evolution. In: IEEE Internation Conference on Web Services, Work In Progress (ICWS 2014 WIP). IEEE, Anchorage, Alaska, USA, pp. 716–719.

Fokaefs, M., Stroulia, E., 2014b. WSDarwin: Studying the Evolution of Web Service Systems. Advanced Web Services. Springer, Ch. 9, pp. 199–223.

Fokaefs, M., Stroulia, E., Messinger, P. R., 2013. Software Evolution in the Presence of Externalities: A Game-Theoretic Approach. In: Mistrik, I., Bahsoon, R., Kazman, R., Sullivan, K., Zhang, Y. (Eds.), Economics-Driven Software Architecture. Elsevier, Ch. 11, pp. 243–258.

Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., 1999. Refactoring Improving the Design of Existing Code. Addison Wesley, Boston, MA.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., Nov. 1994. Design Patterns: Elements of Reusable Object-Oriented Software, 1st Edition. Addison-Wesley Professional.

Gebhart, M., Baumgartner, M., Oehlert, S., Blersch, M., Abeck, S., 2010. Evaluation of service designs based on soaml. In: International Conference on Software Engineering Advances (ICSEA 2010). ICSEA '10. pp. 7–13.

Gokhan, N. M., Needy, N., December 2010. Development of a simultaneous design for supply chain process for the optimization of the product design and supply chain configuration problem. Engineering Management Journal 22 (4), 20–30.

Hadley, M., August 2009. Web application description language. `http://www.w3.org/Submission/wadl/`.

Harsanyi, J. C., 1968. Games with incomplete information played by "bayesian" players, i-iii. part iii. the basic probability distribution of the game. Management Science 14 (7), pp. 486–502.

Hoffmann, W. H., Aug. 2007. Strategies for managing a portfolio of alliances. Strategic Management Journal 28 (8), 827–856.

House, C., August 2012. How restful is your api? `http://www.bitnative.com/2012/08/26/how-restful-is-your-api/`.

Jegadeesan, H., Balasubramaniam, S., 2008. An MOF2-based Services Metamodel. Journal of Object Technology 7 (8), 71–96.

Jersey, September 2014. Restful web services in java. `https://jersey.java.net/`.

Kaminski, P., Litoiu, M., Müller, H., Oct. 2006. A design technique for evolving web services. In: Conference of the Center for Advanced Studies on Collaborative research (CASCON 2006). ACM Press, New York, New York, USA, p. 23.

Karner, G., 1993. Resource estimation for objectory projects. Objective Systems SF AB 17.

Kelter, U., Wehren, J., Niere, J., 2005. A Generic Difference Algorithm for UML Models. Software Engineering 2005, Fachtagung des GI-Fachbereichs Softwaretechnik, 105–116.

Kruchten, P., 2004. The rational unified process: an introduction. Addison-Wesley Professional.

Laffont, J., 2008. externalities. In: Durlauf, S. N., Blume, L. E. (Eds.), The New Palgrave Dictionary of Economics. Palgrave Macmillan, Basingstoke.

Lanza, M., Marinescu, R., Ducasse, S., 2006. Object-oriented metrics in practice. Springer.

Levenshtein, V. I., February 1966. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10 (8), 707–710.

Li, J., Sikora, R., Shaw, M. J., Woo Tan, G., Oct. 2006. A strategic analysis of inter organizational information sharing. Decision Support Systems 42 (1), 251–266.

Li, J., Xiong, Y., Liu, X., Zhang, L., 2013. How does web service api evolution affect clients? In: IEEE International Conference on Web Services (ICWS 2013). IEEE, pp. 300–307.

Lipsey, R. G., Lancaster, K., 1956. The general theory of second best. The review of economic studies, 11–32.

Lyons, K., Messinger, P. R., Niu, R. H., Stroulia, E., Mar. 2012. A tale of two pricing systems for services. Information Systems and E-Business Management 10 (1), 19–42.

Lyons, K., Playford, C., Messinger, P., Niu, R., Stroulia, E., 2009. Business models in emerging online services. In: Nelson, M., Shaw, M., Strader, T. (Eds.), Value Creation in E-Business Management. Vol. 36 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, pp. 44–55.

Maciaszek, L., 2007. Requirements analysis and system design. Pearson Education.

McGuire, T. W., Staelin, R., Jan. 2008. An industry equilibrium analysis of downstream vertical integration. Marketing Science 27 (1), 115–130.

McKendrick, J., November 2011. Cloud Computing's Vendor Lock-In Problem: Why the Industry is Taking a Step Backward. http://www.forbes.com/sites/joemckendrick/2011/11/20/cloud-computings-vendor-lock-in-problem-why-the-industry-is-taking-a-step-backwards/.

Metzger, A., Di Nitto, E., Nov. 2012. Addressing Highly Dynamic Changes in Service-Oriented Systems: Towards Agile Evolution and Adaptation. In: Wang, X., Ali, N., Ramos, I., Vidgen, R. (Eds.), Agile and Lean Service-Oriented Development. IGI Global, Ch. 2, pp. 33–46.

Mikhaiel, R., Lin, G., Stroulia, E., 2006. Simplicity in RNA Secondary Structure Alignment: Towards biologically plausible alignments. In: IEEE Symposium on Bioinformatics and Bioengineering. pp. 149–158.

Mikhaiel, R., Stroulia, E., 2006. Examining Usage Protocols for Service Discovery. In: International Conference on Service Oriented Computing (ICSOC 2006). pp. 496–502.

Monsoon Stone Edge User Forum, June 2011. Amazon soap being discontinued. http://www.stoneedge.net/forum/pop_printer_friendly.asp?TOPIC_ID=12687.

Morgan, R. M., Hunt, S. D., Jul. 1994. The Commitment-Trust Theory of Relationship Marketing. Journal of Marketing 58 (3), 20.

Nagurney, A., 2006. Supply Chain Network Economics: Dynamics of Prices, Flows, and Profits. Edward Elgar Publishing.

Nash, J. F., 1950. Equilibrium points in n-person games. Proceedings of the national academy of sciences 36 (1), 48–49.

Oberle, D., September 2011. D1 report on landscapes of existing service description efforts. http://www.w3.org/2005/Incubator/usdl/wiki/D1.

Oliver, R. K., Webber, M. D., 1992. Supply-chain management: logistics catches up with strategy. Logistics: The Strategic Issues.

Oracle, 2013. wadl2java Tool Documentation. https://wadl.java.net/wadl2java.html.

Ortiz, G., Hernandez, J., 2007. A case study on integrating extra-functional properties in web service model-driven development. In: International Conference on Internet and Web Applications and Services. pp. 35–40.

Ozkaya, I., Kazman, R., Klein, M., May 2007. Quality-Attribute Based Economic Valuation of Architectural Patterns. In: International Workshop on the Economics of Software and Computation. IEEE, pp. 5–5.

Parnas, D. L., Dec. 1972. On the criteria to be used in decomposing systems into modules. Commun. ACM 15 (12), 1053–1058.

Pasquale, L., Laredo, J., Ludwig, H., Bhattacharya, K., Wassermann, B., 2009. Distributed cross-domain configuration management. In: ICSOC-ServiceWave '09. pp. 622–636.

Pautasso, C., Zimmermann, O., Leymann, F., 2008. Restful web services vs. big'web services: making the right architectural decision. In: International Conference on World Wide Web (WWW 2008). ACM, pp. 805–814.

Ponnekanti, S., Fox, A., 2003. Application-service interoperation without standardized service interfaces. In: IEEE International Conference on Pervasive Computing and Communications (PerCom 2003). IEEE Comput. Soc, pp. 30–37.

Ponnekanti, S. R., Fox, A., 2004. Interoperability among independently evolving web services. In: Middleware '04. Springer-Verlag New York, Inc., New York, NY, USA, pp. 331–351.

Roberto Chinnici, Jean-Jacques Moreau, A. R., Weerawarana, S., June 2007. Web services description language (wsdl) version 2.0 part 1: Core language. http://www.w3.org/TR/wsdl20/.

RottenTomatoes, 2014. API Overview. http://developer.rottentomatoes.com/docs.

Ryu, S. H., Casati, F., Skogsrud, H., Benatallah, B., Saint-Paul, R., 2008. Supporting the Dynamic Evolution of Web Service Protocols in Service-Oriented Architectures. ACM Transactions on the Web 2 (2), 1–46.

Seacord, R. C., Plakosh, D., Lewis, G. A., 2003. Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Shoham, Y., Leyton-Brown, K., Dec. 2008. Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press. URL http://www.worldcat.org/isbn/0521899435

Smartbear, 2014. Soapui. http://www.soapui.org/.

Srivastava, A., Sorenson, P. G., 2010. Service selection based on customer rating of quality of service attributes. In: IEEE International Conference on Web Services (ICWS 2010). IEEE Computer Society, Los Alamitos, CA, USA, pp. 1–8.

Staikopoulos, A., Bordbar, B., 2005. A comparative study of metamodel integration and interoperability in uml and web services. In: European conference on Model Driven Architecture: foundations and Applications. pp. 145–159.

Steiner, T., 2007. Automatic multi language program library generation for rest apis. Master's thesis, Institute for Algorithms and Cognitive Systems, University of Karlsruhe.

Swanson, E. B., 1976. The dimensions of maintenance. In: International Conference on Software Engineering. ICSE '76. IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 492–497.

Symons, C. R., 1988. Function point analysis: difficulties and improvements. IEEE Transactions on Software Engineering 14 (1), 2–11.

Tansey, B., 2008. Valuing software services: The real options-based modularity analysis framework. Ph.D. thesis, University of Alberta (Canada).

Tholomé, E., August 2009. A well earned retirement for the soap search api. http://googlecode.blogspot.ca/2009/08/well-earned-retirement-for-soap-search.html.

Trachtenberg, A., October 2003. Php web services without soap. http://www.onlamp.com/pub/a/php/2003/10/30/amazon_rest.html.

Treiber, M., Truong, H.-L., Dustdar, S., 2008. Semf - service evolution management framework. In: Euromicro Conference Software Engineering and Advanced Applications. SEAA '08. pp. 329–336.

Tumblr, 2014. Tumblr API. https://www.tumblr.com/docs/en/api/v2.

Twitter, 2014. REST APIs. https://dev.twitter.com/rest/public.

UDDI Consortium, Nov. 2001. UDDI Executive White Paper. http://uddi.org/pubs/UDDI_Executive_White_Paper.pdf.

Villegas, N. M., Müller, H. A., Tamura, G., Duchien, L., Casallas, R., 2011. A framework for evaluating quality-driven self-adaptive software systems. In: International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011). ACM, New York, NY, USA, pp. 80–89.

W3C, 2001. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

W3C, 2007a. SOAP Version 1.2. http://www.w3.org/TR/soap/.

W3C, 2007b. SOAP Version 1.2 Part 0: Primer (Second Edition). http://www.w3.org/TR/2007/REC-soap12-part0-20070427/.

W3C, 2007c. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

W3C, 2013. Web of Services. http://www.w3.org/standards/webofservices/.

Wang, S., Capretz, M. A. M., 2009. A Dependency Impact Analysis Model for Web Services Evolution. In: IEEE International Conference on Web Services (ICWS 2009). pp. 359–365.

Wang, S., Keivanloo, I., Zou, Y., 2014. How do developers react to restful api evolution? In: Service-Oriented Computing. Springer, pp. 245–259.

Xing, Z., 2010. Model Comparison with GenericDiff. In: 25th IEEE/ACM International Conference on Automated Software Engineering. pp. 135–138.

Xing, Z., Stroulia, E., 2005a. Analyzing the Evolutionary History of the Logical Design of Object-Oriented Software. IEEE Transactions on Software Engineering 31 (10), 850–868.

Xing, Z., Stroulia, E., 2005b. UMLDiff: An Algorithm for Object-Oriented Design Differencing. In: 20th IEEE/ACM International Conference on Automated Software Engineering. pp. 54–65.

Xing, Z., Stroulia, E., 2006. Refactoring Detection based on UMLDiff Change-Facts Queries. In: 13th Working Conference on Reverse Engineering. pp. 263–274.

Zhang, K., Stgatman, R., Shasha, D., 1989. Simple fast algorithm for the editing distance between trees and related problems. SIAM Journal on Computing 18, 1245–1262.