COMPOSING HETEROGENEOUS SERVICES FROM END USERS' PERSPECTIVE

by

Bipin Upadhyaya

A thesis submitted to the
Department of Electrical & Computer Engineering
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
June 2014

Abstract

As the Internet becomes more pervasive, the content and services are increasing in quantity as well as improving in quality. This trend is fostered by the advancement of technologies, such as RESTful services, Web 2.0, and Mashups. Service composition integrates services to fulfill specific tasks using a set of tools. The existing service composition techniques and tools are mainly designed for the Service Oriented Architecture (SOA) professionals. The business processes used in the service composition systems are primarily designed by business analysts who have extensive process knowledge. Due to the lack of process knowledge, novice business analysts and end users face challenge to identify and orchestrate service into a well-defined business process. Even for the experienced users, it is challenging to select appropriate services from a set of functionally similar services as the quality information of services may not be available.

In this thesis, we propose a framework that allows a non-technical user to combine web services to achieve a goal. Our approach helps users to find the process knowledge from the web. We index web services based on the semantic concepts available in the service description documents and help users to formulate a web service search query. We use online reviews to choose a web service from a set of functionally similar web services. Our approach automatically finds the data flow between web services and

generates a user interface to execute a composite service. The effectiveness of our proposed approaches is demonstrated through a series of case studies. The results of our case studies show that our approaches for process knowledge extraction, service discovery, and service selection make it easier for people with less technical knowledge to compose services.

Statement of Co-Authorship

The content of this thesis has been published in the following papers. More specifically, Chapter 1 is based on paper [1]. Chapter 3 is based on papers [2] and [3]. Chapter 4 is based on paper [7]. Chapter 5 is based on papers [8] and [9]. Chapter 6 is based on papers [4], [5] and [6].

1. B. Upadhyaya, and Y. Zou, Integrating Heterogeneous Web Services from an End User Perspective, ACM/IFIP/USENIX International Middleware Conference, Doctoral Symposium, December 3-7, 2012, Montreal, Canada

2. B. Upadhyaya, Y. Zou, H. Xiao, J. Ng, and A. Lau, Migration of SOAP-based Services to RESTful Services, International IEEE Symposium on Web Systems Evolution (WSE), pages 105-114, September 30, 2011, Williamsburg, VA, USA

3. B. Upadhyaya, F. Khomh, Y. Zou, Extracting RESTful Services from Web Applications, 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pages 1-4, December 17 - 19, 2012, Taipei, Taiwan

4. B. Upadhyaya, R. Tang, Y. Zou, An Approach for Mining Web Service Composition Patterns from Execution Logs, Journal of Software: Evolution and Process, Wiley, Volume 25, Issue 8, pages 841870, August 2013

5. B. Upadhyaya, H. Xiao, Y. Zou, J. Ng, and A. Lau, A Framework for Composing Personalized Web Resources, A Framework for Composing Personalized Web Resources, In Lecture Notes in Computer Science (LNCS), pages 56-86, Springer-Verlag 2013

6. B. Upadhyaya, Y. Zou, S. Wang, J. Ng, An Automatic Approach for Service Composition by Mining Process Knowledge from the Web, 11th International Conference on Service Oriented Computing (ICSOC 2013), pages 267-282, December 2-5, 2013 - Berlin, Germany. Springer

7. B. Upadhyaya, F. Khomh, Y. Zou, A. Lau, and J. Ng, A Concept Analysis Approach for Guiding Users in Service Discovery, 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA), pages 1-8, December 17 - 19, 2012, Taipei, Taiwan

8. B. Upadhyaya, Y. Zou, I. Keivanloo, and J. Ng, Quality of Experience: What End-users Say About Web Services?, 21th IEEE International Conference on Web Services (ICWS), June 27 - July 2, 2014, Alaska, USA. IEEE.

9. B. Upadhyaya, Y. Zou, J. Ng, T. Ng, and D. Lau, Towards Quality of Experience driven Service Composition, 2nd International Workshop on Personalized Web Tasking (PWT 2014) on IEEE Services, June 27 - July 2, 2014, Alaska, USA. IEEE.

The aforementioned papers resulting from this thesis were co-authored with my supervisor Dr. Ying Zou, researchers (Ms. Joanna Ng, and Mr. Alex Lau) at IBM Toronto Lab, and colleagues (Dr. Foutse Khomh, Dr. Iman Keivanloo, Mr. Ran Tang, Dr. Hua Xiao, Mr. Shoahua Wang) in our lab. In all cases, I am the primary

author. More specifically, Dr. Ying Zou supervised all the research related to those papers. Ms. Joanna Ng, Dr. Foutse Khomh, Dr. Iman Keivanloo, Mr. Alex Lau, Ms. Tinny Ng, Ms. Diana Lau and Mr. Hua Xiao participated the discussion meeting of the research projects related the papers which they are co-authors and gave feedback as well as suggestions to improve the research. The paper "An Approach for Mining Web Service Composition Patterns from Execution Logs", Mr. Ran Tang contributed the content related to business process mining which is not a part of this thesis. My contribution to the paper was identifying the data flow and the control flow between services and extending the case study by introducing web services, which is included in this thesis.

Acknowledgments

These past four years have been extremely exciting and furiously challenging at the same time. During these four years, I learned the most valuable lessons, not just those found in the text books, but also valuable life lessons. Many people helped me acquire these experiences and helped me to be here at this important juncture of my life. I would like to extend my heartfelt gratitude and appreciation to all of them.

First and foremost, I would like to extend my heartiest gratitude to my adviser, Dr. Ying (Jenny) Zou. I was fortunate to have a mentor like her who was always available and willing to help and support me; not only when I was doing good but also when I was facing setbacks. She was always there to guide, encourage and inspire me during both the smooth times and the rough times. Her creativity, knowledge and experience made so many difficult things possible; including this dissertation. I am also grateful to my committee members: Dr. Thomas R. Dean, Dr. Patrick Martin, Dr. Scott Yam and Dr. Yuhong Yan for their efforts and feedback.

I would like to thank my colleagues Dr. Hua Xiao, Mr. Shoahua Wang, Mr. Ran Tang, Dr. Iman Keivanloo, and Dr. Foutse Khomh for their creative inputs. I also want to thank Prabeen Joshi, another colleague, who provided valuable assistance in getting my early days in Kingston going. Many thanks to Mr. Tejinder Dhaliwal, Mr. Dwaipayan Sinha, Mr. Hao Yuan, Mrs. Lilian Barbour, Mr. Ehsan Salamati, Mr.

Feng Zang and Ms. Haoran Niu for sharing so many memorable moments with me during the long hours of development and testing work in the Software Re-engineering lab. I would also like to thank Ms. Joanna Ng, Mr. Alex Lau, Ms. Tinny Ng and Ms. Diana Lau at IBM Toronto Lab for their valuable suggestion and inputs to shape my research. Special thanks to Ms. Debra Fraser to helping me with the different issues during the graduate studies at Queen's University.

At this moment, I would also like to recognize and appreciate the unconditional love and sacrifice of my late father H.P. Upadhyaya, my mother Pushpa Upadhyaya, my two brothers who have always been highly supportive of my never ending quest for higher studies. Specially, I want to dedicate this dissertation to my father, who had been the primary source of inspiration, strength and motivation of my life.

Last but, not the least, special thanks to my wife Anuja for her infinite love, patience, encouragement and support during these long and monotonous years for her while I have been dealing with the instability of graduate studies.

Sincerely,

Bipin Upadhyaya

Kingston, Ontario

Statement of Originality

I hereby certify that all of the work described within this thesis is the original work of the author. Any published (or unpublished) ideas and/or techniques from the work of others are fully acknowledged in accordance with the standard referencing practices.

Bipin Upadhyaya

April, 2014

Contents

List of Figures

List of Tables

Glossary

BPEL Business Process Execution Language.

CSS Cascading Style Sheets.

CTT Concur Task Trees.

DAML-S DARPA Agent Markup Language for Services.

DOM Document Object Model.

GOMS Goals, Operators, Methods, and Selection .

GUI Graphical User Interface.

HATEOAS Hypermedia as the Engine of Application State.

HTA Hierarchical Task Analysis.

HTML HyperText Markup Language.

HTTP HyperText Transfer Protocol.

IR Information Retrieval.

JS JavaScript.

MOF Meta-Object Facility.

NLP Natural Language Processing.

OWL-S Web Ontology Language for Services.

POS Part Of Speech.

QoE Quality of Experience.

QoS Quality of Service.

REST Representation State Transfer.

SMART Service-Oriented Migration and Reuse Technique.

SOA Service Oriented Architecture.

SOAP Simple Object Access Protocol.

SQL Structured Query Language.

SVD Singular Value Decomposition.

TF-IDF Term Frequency - Inverse Document Frequency.

UI User Interface.

WADL Web Application Description Language.

WRDL Web Resource Description Language.

WSDL Web Service Description Language.

WSFL Web Service Flow Language.

WSMO Web Service Modeling Ontology.

XML Extensible Markup Language.

Chapter 1

Introduction

Software is common in all aspects of our lives, such as checking a stock price, finding a doctor, and buying a product. Nowadays, a significant part of any software system is structured using software services and implemented using web service technologies. The web has become a basic infrastructure to support a user to search and perform different tasks, such as buying movie tickets and making a reservation. A user employs a web browser to perform a task. With increasing information sources and services, it is difficult for a user to find the desired services and integrate these services to perform a task. Programmable Web [100] alone has indexed more than 9000 services that are used in our daily activities, such a, dating, shopping, and job searching. However, a single service cannot fulfill a user's intent. For example, if a user wants to plan a trip, he may need a flight booking and a hotel reservation services. One or more services are combined to fulfill his purpose.

Service composition is a process that combines a number of logically related services to achieve a given goal. In the current state of practice, the integration approaches are based on a predefined process model that describes the services to perform a task, such as planning a trip. A service performs a specific job, such as finding

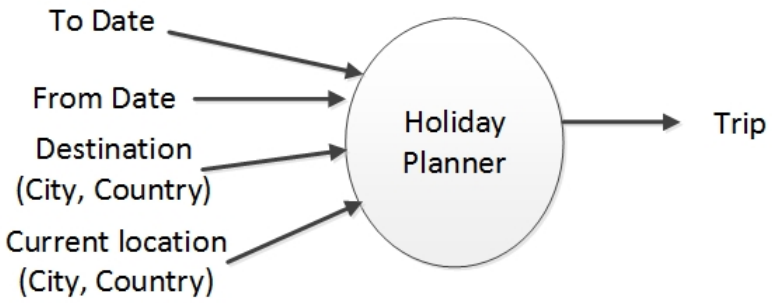


Figure 1.1: Abstract specification of holiday planning

the availability of a room. Imagine a scenario, such as planning a holiday that includes both buying a flight ticket and booking a hotel room. A user's decision to choose a destination may be based on the availability of both flight tickets and hotels. A user has to check the availability of a hotel before buying a flight ticket and correspondingly check the availability of a flight ticket before booking a hotel. A user has to switch between different web sites to perform a basic task. The situation becomes more complex and difficult when a user has to perform a task dependent on different cases. Service composition involves the development of customized services by discovering, integrating, and executing existing services. Most of the available web services are fine grained and provide specific sets of functionalities. The fine granularity of functionality gives the freedom to the user to embrace different kinds of services. For instance, a holiday-booking application from a user's perspective may consist of a flight-booking service, a hotel-booking service, and a local events service. The same holiday-booking application from another user perspective may not contain exactly the same set of services. Users have their own choices and preferences. Hence, users should be allowed to expand and customize different services. There are numerous service providers offering the equivalent kinds of services.

Figure 1.1 shows an abstract definition of a holiday planner. A holiday planner is a black box containing services to book a flight, reserve a hotel, and get information about related activities to perform on a holiday. It is challenging to find an atomic web service that can take these exact inputs, produce the desired output, and satisfy any user constraints outlined in the task description. Therefore, we need to take individual web services and compose services together in a way that the intent of planning a holiday is met. This requires decomposing an abstract requirement into abstract subtasks that eventually equate to web services. The subtasks are connected through some process logic.

For example, Figure 1.2 can be thought of as an abstract composition derived from the high-level abstract definition of Figure 1.1. The solid circles represent individual tasks, dotted circles represent intermediate states of services and the arcs represent data flow among the services. When each task is sufficiently simplified into subtasks by decomposing a higher-level specification and constraints, atomic web services are bound to the individual tasks and invoked. A user combines several fine-grained services to deliver a high granular service as in the case of the holiday planner in Figure 1.2. For example, if a user has a relative or a friend at the destination, he can arrange the holiday and remove the hotel booking service. The service composition needs to find the appropriate services based on the task described in a natural language. Service composition needs to find the data flow between the services. Data flow is available between the confirmed flight service and the book-hotel service. In the example shown in Figure 1.2, the hotel-booking service needs to know the time when the flight lands. The control flow is essential since it defines the order in which the services are invoked.

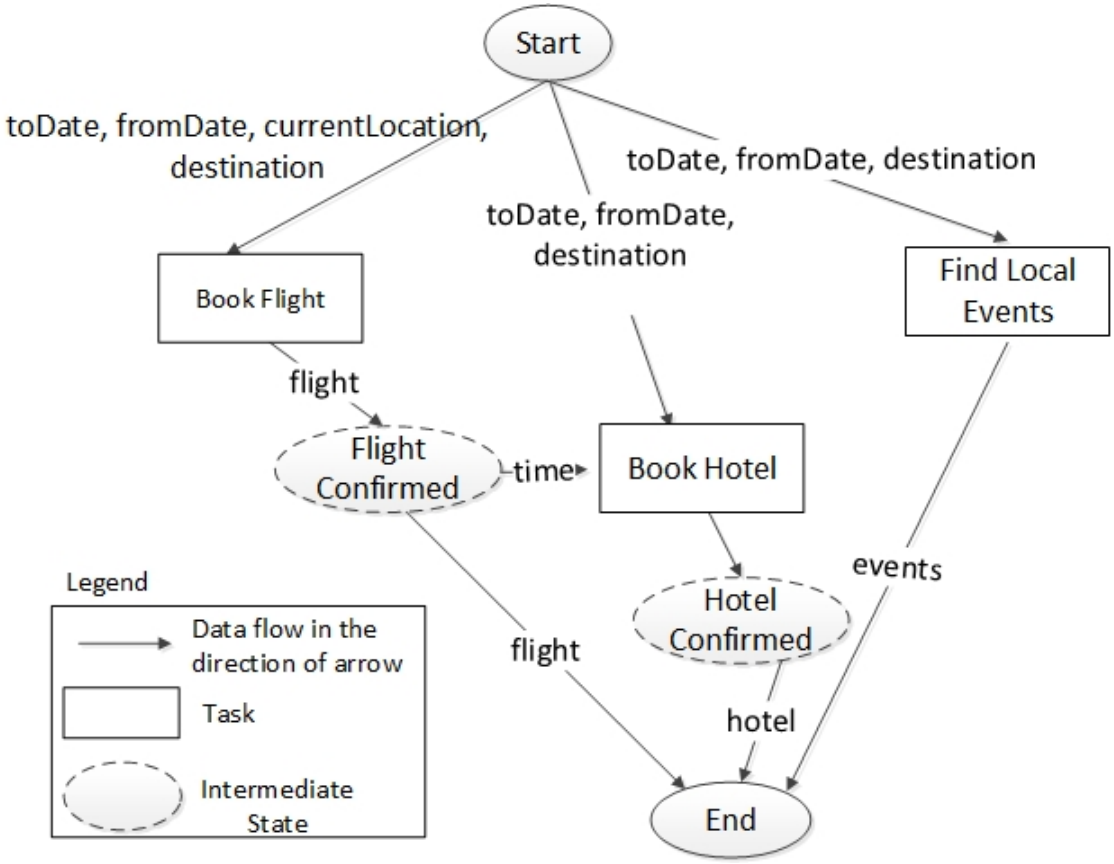


Figure 1.2: Abstract service composition showing different tasks invoked in holiday planning

1.1 Steps in Service Composition

A service provider advertises the service description documents in a web service registry. The web service registry contains different kinds of service descriptions, such as Web Service Description Language (WSDL) [34] and Web Application Description Language (WADL) [127]. A service customer queries the registry to retrieve services. A service consumer uses a natural language to query the registry. There may be more than one service; the service selection may depend on user preferences and historical information. Service descriptions specify capabilities of services and

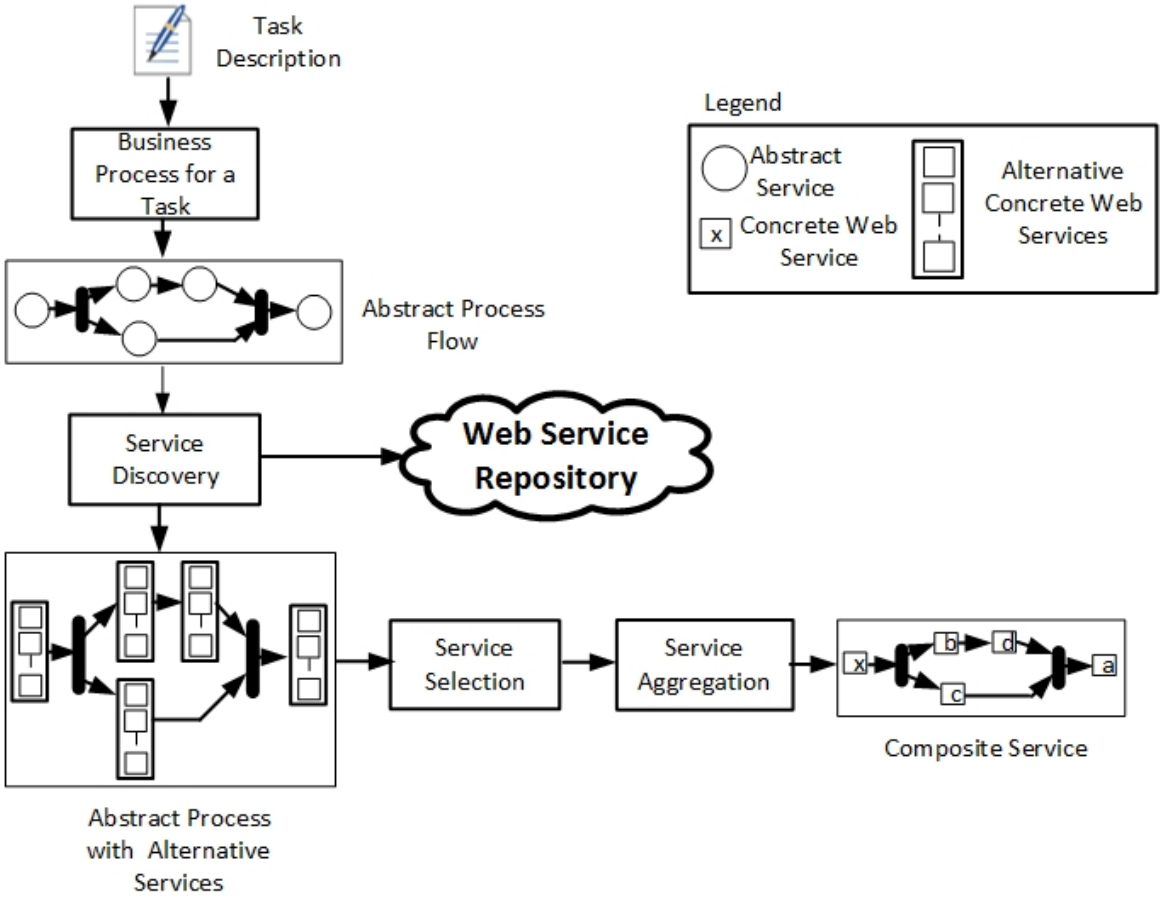


Figure 1.3: Typical steps involved in a service composition process

usually include inputs, outputs, exceptions, functional and non-functional description. Different metrics need to be considered when selecting a service (*e.g.*, trust of the service, non-functional properties, and input/output compatibility). Service selection may require service customers' input. Once all the related services are selected, the flow of data between the services can be either automatically generated using planning algorithms or defined by a user. Figure 1.3 shows the steps involved in service composition which are described below.

Business Process for a Task

A task usually cannot be fulfilled by a single service. We need to integrate various services using business processes. Standards, such as Business Process Execution Language (BPEL) [136], are used to specify formal process models. A business process has a number of logically related tasks to achieve a business objective. A process defines tasks, connections, roles, and resources. Tasks define the operations for achieving business objectives. Connections include control flow and data flow among tasks. Process knowledge is essential for service composition systems to generate business processes or ad-hoc processes. The process knowledge mostly comes from business analysts, service descriptions, business requirements, and process description documents.

Service Discovery

Service discovery is a process of finding services based on user requirements. Typically, a service discovery involves four steps: a) service providers advertise the capabilities of a service using service description documents (*e.g.*, WSDL [34] and WADL [127]) and register the service in a service repository; b) a middle agent crawls different service repositories and stores service description documents locally; c) a service requester queries the middle agent for the service providers that can best match the desired capabilities of services, and d) the middle agent tries to match the request against the stored advertisements and returns the matched result.

Service Selection

Service selection is the process of finding a suitable service from a pool of functionally equivalent services. Most of the research in service selection is based on Quality of Services (QoS) [3, 28, 147], either by proposing a comprehensive QoS language to describe service requests and offers, or implementing a selection algorithm to achieve an optimized result. Generally, users express their preferences using a QoS-based language [59]. QoS-based approaches help a user choose a service from the results of functionally equivalent services. Reputation and trust-based mechanisms [75, 80] are also used to filter best services from a pool of functionally equivalent services.

Service Aggregation

Composite services are recursively defined as an aggregation of elementary and composite services. The service composition allows a user to create applications on top of service description, discovery, and communication capabilities [3, 17]. Composite services offer reusable capabilities to developers. Service composition provides a seamless access to a variety of complex services to a user. Control flow information, which gives the order of execution of services, is available in the process knowledge. In this stage, we perform input/output data matching also called data flow. Data flow helps to identify reusable data inputs among services, helping the user to avoid multiple inputs.

1.2 Research Challenges

Recent advancement in web services makes it practically possible to publish, locate, and invoke services across the web. However, there are different formats and standards

to describe service interfaces. Heterogeneous service description documents are difficult for a user to understand. With the ever-increasing number of services published on the Internet (*e.g.*, Google [48] has indexed 167,000 WSDL documents), finding the desired services is just similar to looking for a needle in a haystack. In some cases, if no single service can satisfy the functionality required by the user, there should be a possibility to integrate existing services together to fulfill the request. The problem of service composition is a highly complex task. Here, we highlight some sources of its complexities:

- **Heterogeneity in service and service description languages.** Web services developed by different organizations use different models to describe services. There are mainly two different types of services (*i.e.*, SOAP-based services and RESTful services). For each type of services, there are several service description languages. It is difficult for users to understand and parse different service description languages.

- **Semantic gap between service providers and users.** The vocabulary adopted in a service description document is often used by developers in the software development domain. It can be very different from the ones used by the user in search queries. The semantic gap between service providers and users make the service discovery difficult and challenging. Web service search engines provide little support for users to construct and reformulate their queries when the initial query fails. The available web service discovery approaches act as a black box to users. A user has to perform several trials in order to formulate an appropriate query to retrieve the desired services.

- **Lack of QoS information for service selection.** Current approaches for

service selection (*e.g.*, [21, 22, 102, 145, 146]) are based on the non-functional aspects of web services. However, the process of obtaining QoS information is largely overlooked. The static QoS release is not frequently updated and is done in a specific environment and platform. The posted QoS information may be different if the same service invoked from a different geographical location or through a different device. Hence, the static information is less reliable. Runtime monitoring to collect QoS of web services at client side is resource intensive, time consuming, and expensive [4]. This issue threatens the applicability of QoS-based service selection approaches (*e.g.*, [75, 80]).

- **Lack of process knowledge to link and execute services.** Process knowledge integrates different services to implement business processes. In the current state of practice, business processes are primarily designed by expert business analysts who have extensive process knowledge. It is challenging for novice business analysts and end-users to identify services to orchestrate a well-defined business process due to the lack of process knowledge. Locating appropriate services and linking the identified services are challenging even for experienced developers. Current work in service flow identification [23, 68, 125] does not help to identify tasks as those methods are solely based on input and output parameters of services. A user communicates with a task through a user interface (UI). Current approaches in the UI generation for web services [62, 63] are based on technical descriptions and therefore, are difficult to understand and error prone.

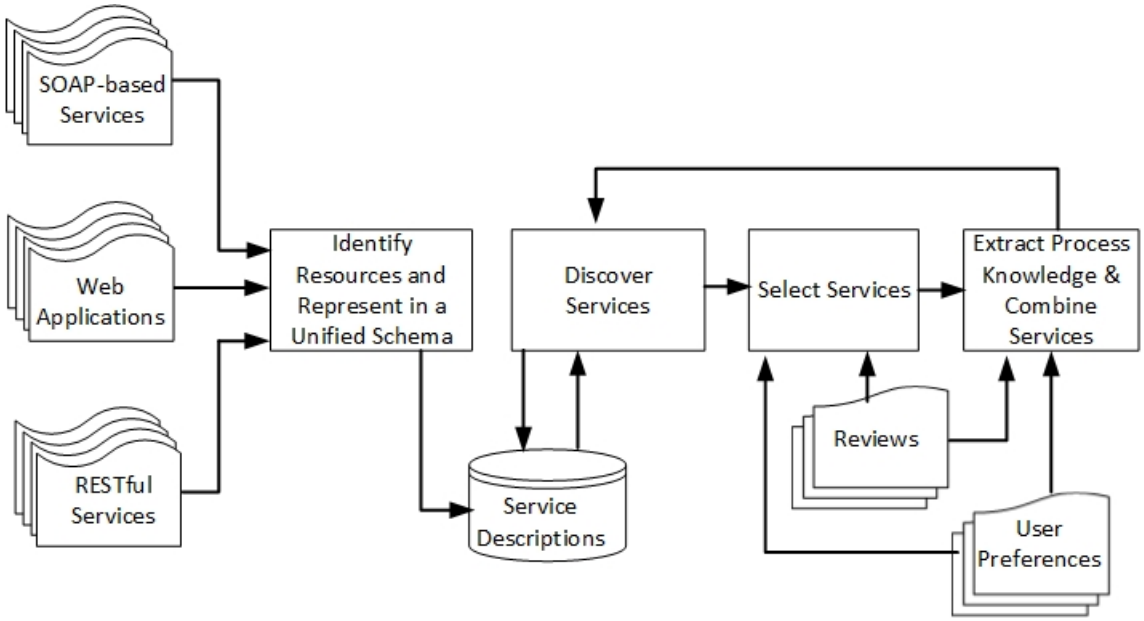


Figure 1.4: Overview of our proposed framework

1.3 Thesis Objectives

Figure 1.4 shows the overview of this thesis. As shown in 1.4, we identify RESTful services from SOAP-based services and web applications. Based on a user's goal, we find process knowledge, select services and combine services to create a composite service as shown in Figure 1.4. Since process knowledge is difficult to obtain and understand, service composition is not accessible for end users. Our work on process knowledge makes service composition possible to end users. Moreover, our approach helps to rank services in absence of quality of service (QoS) information. In the Figure 1.4, a manual verification of extracted services is required in the "Identify Resources and Represent in a Unified Schema" step. All other process are automated. To address the limitations outlined in section 1.2, we discuss the possible solutions.

Identify RESTful services from SOAP-based services and Web applications. To facilitate the usage of heterogeneity in services and service description languages, we identify RESTful services from SOAP-based services and web applications. Encoding and decoding of XML-based SOAP messages consume resources (*i.e.*, battery, processor speed, and memory). SOAP-based services ignore the semantics of the operations of underlying protocols, whereas web applications are mainly designed for human users. Migration of SOAP-based services to RESTful services makes the services more pervasive, faster, and suitable for thin clients. Similarly, our framework identifies resources from web applications. Information sources and services available in the web may not be directly used for service composition as they may be implemented using different protocols (*e.g.*, SOAP, HTTP, and RPC) and formats (*e.g.*, WSDL and WADL). Unifying different formats and providing a global view of service make the service discovery and linking easier.

Extract semantic concepts from service descriptions and help a user to formulate queries for services. Web service descriptions contain technical information and are hard to understand. To bridge the semantic gap and query formulation we use natural language processing techniques to extract concepts present in the service description documents. These concepts are the basis for service linkage and service discovery. A web service search query results in more than one related services. We present an approach that helps a user to craft the web service search query. Our approach does not require a user to learn any new query language.

Extract quality information from the web. User oriented content generation approach of Web 2.0 has enabled people to broadcast their knowledge and experience to the masses. Online user review is an example of such a phenomenon. End-users

express their experiences via online reviews to reveal their satisfactions and disappointments about services. We consider the possibility of exploiting user reviews for service selection applications. We propose the concept of quality of experience (QoE) which measures a customer's satisfaction with a service. Extracting QoE attributes from user reviews is challenging. User reviews are written in natural language and presented as unstructured data. Therefore, it is not trivial for computers to understand, analyze, and aggregate QoE from the web. We present an approach to extract QoE attributes and use sentiment analysis to understand the users' perspective on the extracted QoE attributes.

Extract process knowledge from the web and link services. A user has to repeatedly search the web to learn and complete different tasks required to achieve a goal. Knowledge from business processes to describe the tasks for achieving a goal, if available, is hard for a novice designer or a user to understand. To address the challenge on the lack of process knowledge for end users to link and execute services, we present an approach that automatically extracts process knowledge for a task from the web. Our approach helps a user to discover web services and establish control flow between the selected web services to perform a task. The relationship between different web services explains how a service can be invoked. Our approach identifies and converts data format between different resources. Moreover, our approach automatically creates a GUI based on service linkage to execute a task.

1.4 Research Statement

Process knowledge is critical for service composition. In the current state of practice, service compositions are designed by business analysts and developers with extensive

process knowledge. It is challenging for novice business analysts and end users to identify a complete set of services to orchestrate a well-defined service composition due to lack of process knowledge. This thesis investigates problems during different stages of service composition such as service discovery, service selection and service linkage. We build an approach to extract process knowledge from the web to help users compose services. Our approach identifies quality information from online reviews to rank services. We recognize the data flow and the control flow between services. We generate composite services using the extracted process knowledge and the flow information between services. We relate users' goals with quality information and use the quality information during the service composition process. Our framework shields non-professional users from the complexity of service composition frameworks.

1.5 Outline of the Thesis

This thesis contains a survey of the different stages of web service composition, a description of our work, and an overview of the results and contributions. An outline of the structure of this thesis is shown as follows:

Chapter 2 reviews the background and related work for supporting users in service composition. We discuss in detail about service oriented architecture and different types of services. We discuss related work in the area of service migration, service discovery and service linkage.

Chapter 3 gives an overview of our approach to identify resources from SOAP-based services and web applications. The identified resources can be used as RESTful services. The chapter presents the proposed unified schema that can represent different kinds of web services. Through a case study, we show the effectiveness of our

approach to identify resources.

Chapter 4 gives an overview of our concept-based service discovery approach. We perform natural language processing on service description files and store the service description based on the concepts available. We provide an approach to support users who query the service repository to find a suitable service. Our approach also links and recommends suitable services, making the service discovery process more efficient and suitable to novice developers and users.

Chapter 5 elaborates the method of service selection. Non-functional qualities, such as response time, and availability are difficult to gather. In our work, we collect users' feedback about a service and develop a method to index service based on reviews posted on websites. The case study demonstrates that reviews are as reliable as traditional quality of services in selecting web services.

Chapter 6 elaborates the method of service composition. We describe our approach to extract process knowledge from the web and then use it to compose services. The case study shows that our approach efficiently extracts process knowledge from the web. We use this process knowledge to select and link services to form composite services.

Chapter 7 concludes the thesis by summarizing our contributions and discussing future research directions.

Chapter 2

Background and Related Work

In this chapter, we present the background of service oriented architecture and survey of related works. Section 2.1 introduces the background of service oriented architecture and compares different types of services. Section 2.2 provides an overview in service migration. Section 2.3 covers different service description models, and Section 2.4 discusses service discovery based on functionality of services. Section 2.5 discusses service selection mechanisms. We describe service composition approaches and frameworks in Section 2.6. Finally, Section 2.7 summarizes the chapter.

2.1 Service Oriented Architecture

Service-Oriented Architecture (SOA) [17] is an architectural paradigm for designing and developing distributed systems. SOA solutions have been created to satisfy business goals that include flexible integration with legacy systems, streamlined business processes, reduced costs, innovative service to customers, and agile adaptation. Though a lot of definitions [7, 17, 39, 40] are available, the core idea of SOA revolves around the notion of service. According to Bianco *et al.* [17], a service has the following common properties:

1. A service is self-contained. A service is highly modular and can be independently deployed.

2. A service is a distributed component. A service is available over the network and accessible through a name or locator other than the absolute network address.

3. A service has a published interface. Users of the service need to see the interface and can be oblivious to implementation details.

4. A service stresses interoperability. Service users and providers can use different implementation languages and platforms.

5. A service is discoverable. A particular directory service allows the service to be registered so users can lookup the required service.

6. A service is dynamically bound. A service user does not need to have the service implementation available at build time; the service is located and bound at run-time.

The aforementioned characteristics describe an ideal service. In reality, services implemented in service-oriented systems lack or relax some of these characteristics, such as being discoverable and dynamically bound.

For SOA, the basic component types are service users and service providers. Auxiliary component types, such as the Enterprise Service Bus (ESB) [40] and directory of services can be used. SOA connector types include synchronous and asynchronous calls using Simple Object Access Protocol (SOAP) [52], bare HTTP [57], and messaging infrastructure. Many properties can be assigned to these components and connector types, but they are usually specific to each implementation technology.

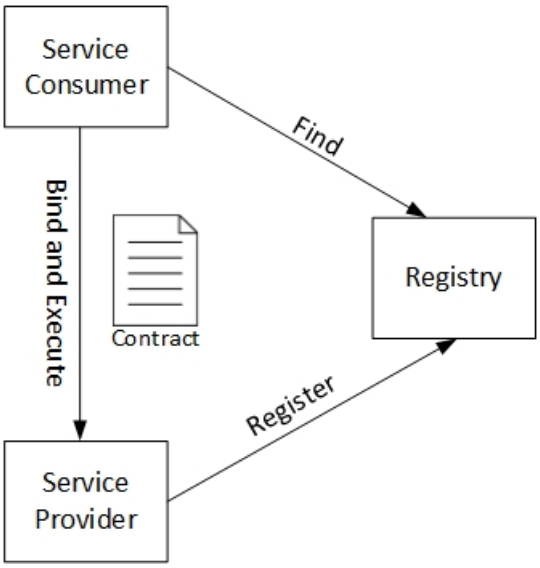


Figure 2.1: Interaction between service consumer and service provider

SOA has three main components: registry, service customer, and service provider. Figure 2.1 shows the interaction among those three components. The service provider advertises the services in the registry. The customer looks for the service and then consumes the services as mentioned in the contract. The constraints that apply to the SOA architectural style are listed as follows:

- **Service Contracts.** Two services must come to an agreement to interact properly [147]. A contract formalizes the details of a service, such as the contents, the price, the delivery process, and the quality criteria. For each interaction between these two parties, one provides some combination of data and functionality and the other consumes it. Before the provider can provide whatever the service offers, however, the two parties must come to an agreement, or contract that specifies the details of the service the provider is performing. A service contract gives a mechanism that can be used to achieve meaningful forward compatibility. Forward compatibility is the ability of a system to gracefully

accept the input intended for later versions of itself. In particular, a service contract specifies the functional and non-functional requirements [94]:

- **Service registry.** A service registry is a network-based directory that contains available services. A service registry accepts and stores contracts from the service providers and provides the contracts to the interested service consumers. A service registry is a bridge between the service providers and the service consumers. A service registry enables the service consumers to access a wide range of web services that matches specific searching criteria. The first public UDDI Business Registry (UBR) [123] nodes were run by IBM, Microsoft, and SAP [106]. Centralized registries can provide an efficient way to manage a certain number of service registrations [28, 29]. However, the centralized registries share the common problems of centralized systems [14], such as, a single point of failure, bottleneck, and scalability. Another approach to organize service registries is to use techniques from distributed systems to manage multiple registries.

- **Service provider.** The service provider is a service or a network-addressable entity that accepts and executes requests from the consumers. It can be a mainframe system, a component, or a software system that executes the service. The service provider publishes the contract of the registry for the service consumers to access.

- **Service consumer.** The service consumer is an application, a service, or some other type of software module that requires a service. A service consumer finds the location of the service in the registry, binds to the service over a network

and executes the service function. The service consumer executes the service by sending a request formatted according to a contract.

2.1.1 Types of Services

In SOA, resources are made available to other participants in the network as independent services that are accessed in a standardized way. Most definitions of SOA identify the use of web services in its implementation. However, it is possible to implement SOA using any service-based technology. We describe two most popular types of service in this section: a) SOAP-based services; and b) services based on REpresentational State Transfer (REST) architecture (also called RESTful services).

SOAP-based Web Services

Web services are commonly used to implement SOAs. Service interfaces are defined in the Web Service Description Language (WSDL) [34], and the service users and the service providers communicate using the SOAP protocol [52]. WSDL describes web services to separate the description of the abstract functionality offered by a service from the concrete details of a service description, such as "how" and "where" that functionality is offered. Two attributes in a WSDL interface, "style" and "use", define the SOAP communication between the service users and the service providers. The style attribute has two possible values: "RPC" and "document". The use attribute refers to a data encoding which also has two possible values: "encoded" or "literal". Consequently, there are four possible combinations of these two attributes. Two combined options that are common in practice are RPC-encoded and document-literal.

A WSDL document defines a service as a collection of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages are separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and the data format specification for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service. Hence, a WSDL document uses the following elements in the definition of network services:

- **Types**: a container for data type definitions using some type system (such as XSD).
- **Message**: an abstract, typed definition of the data being communicated.
- **Operation**: an abstract description of an action supported by the service.
- **Port Type**: an abstract set of operations supported by one or more endpoints.
- **Binding**: a concrete protocol and data format specification for a particular port type.
- **Port**: a single endpoint defined as a combination of a binding and a network address.
- **Service**: a collection of related endpoints.

The services are advertised in Universal Description, Discovery, and Integration (UDDI) [123]. The UDDI specifications offer the users a unified and systematic way

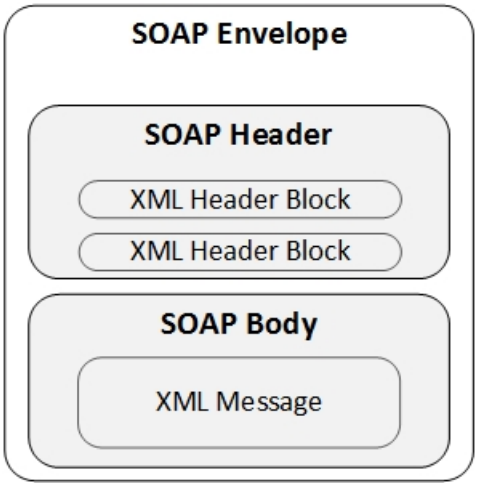


Figure 2.2: SOAP message format

to find the service providers through a centralized registry of services that are roughly equivalent to an automated online "phone directory" of web services. The browser-accessible UDDI Business Registry (UBR) became available shortly after the specification went public. Several individual companies and industry groups are starting to use "private" UDDI directories to integrate and streamline access to their internal services. UDDI provides two basic specifications that define a service registry's structure and operation: 1) a definition of information to provide about each service, and how to encode the information; 2) a query and update API for the registry that describes how this information can be accessed and updated. Registry access is accomplished using a standard SOAP API for both querying and updating. The new repositories such as ProgrammableWeb [100], and XMethods [141] are emerging as public repositories, providing better search option and ranking of services.

SOAP is fundamentally a stateless, one-way message exchange paradigm between two SOAP nodes, from a SOAP sender to a SOAP receiver. By combining one-way

exchanges with the features provided by the underlying transport protocol and/or application specific information, SOAP can be used to create more complex interactions such as request/response, and request/multiple response.

A SOAP message is a standard XML document. Figure 2.2 shows the basic structure of a SOAP message. The root element is an envelope tag, which contains two elements: body and header. The body element provides a simple mechanism for exchanging mandatory information intended for the ultimate recipient of a message. The body element contains an XML document which represents structured return data, arguments, or fault for error reporting. As the SOAP message travels from server to server, tags are read and possibly acted upon. Other tags in the header can be instances of a transaction or session ID tags to create a state, although they can be anything. Thus, SOAP has the flexibility to deal with situations that have not been created or encountered yet.

RESTful Services

REST [42] avoids the complexity of the processing overhead of the SOAP-based services. One notable REST concept is a resource, which is a piece of information that has a unique identifier (*e.g.*, a uniform resource identifier (URI)). It is no coincidence that URIs look like URLs in a web browser. REST relies on the HTTP protocol for the interaction between service users and providers. The HTTP protocol has four basic operations: POST, GET, PUT, and DELETE. A GET request on a URI tells the service provider to retrieve the data. A PUT request indicates that a service provider should update the old data with the data sent in the request. A DELETE request indicates that a service user wants a service provider to delete the data. A

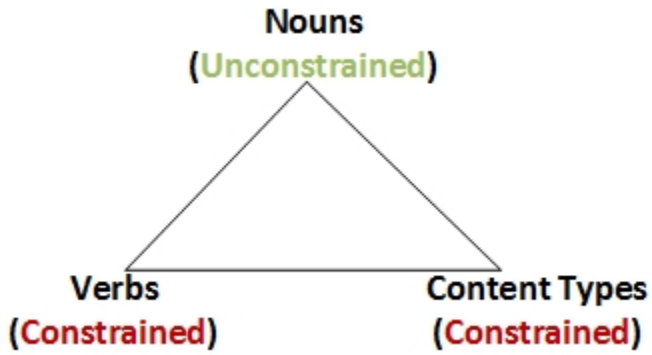


Figure 2.3: REST Triangle

POST request is used to create a new resource. The HTTP protocol defines the status codes that can be returned: 200 for OK, 201 for created, 401 for unauthorized, and so forth. HTTP status code is primarily divided into five groups for better explanation of request and responses between client and server as named: “Informational 1XX”, “Successful 2XX”, “Redirection 3XX”, “Client Error 4XX”, and “Server Error 5XX”. Table 2.1 shows the CRUD and RESTful services. The error messages generated by the RESTful services are mostly errors described by the HTTP protocol. In a REST design, applications of these operations on resource URIs correspond to the CRUD operations; create, retrieve, update, and delete are commonly used in information systems. Figure 2.3 represents a REST triangle: the resources are generally denoted by nouns; the verbs denote the action, and each resource returns a representation in the defined content-type. The end goal is to have resources that have a uniform interface. All resources are addressable, there is a common verb vocabulary for resources, and there is flexible but understood representations of resources. This enables for radical simplification so that data can be easily consumed by any application.

There are six core REST design principles [42]:

Table 2.1: CRUD and HTTP verbs

CRUD operations	**HTTP commands**	**Input formats**	**Output status if successful**
Create	POST	HTTP form encoded	Status 201 created
Read	GET	None	Status 200 OK
Update	PUT	HTTP form encoded	Status 200 OK
Delete	DELETE	None	Status 200 OK

1. The key abstraction of information is a resource, named by an URI. Any information can be named as a resource: a document or image, a temporal service (such as the current temperature in Kingston) or a collection of other resources.

2. The representation of a resource is a sequence of bytes, plus representation meta-data to describe those bytes. REST introduces a layer of indirection between an abstract resource and its concrete representation.

3. All interactions are context-free. REST applications are not necessarily without the state, but each interaction contains all of the information necessary to understand the request.

4. Only a few primitive operations are available. REST components can perform a small set of well-defined methods on a resource to produce a representation.

5. Idempotent operations and representation meta-data are encouraged in support of caching. The meta-data included in requests and responses permit REST components (such as user agents or caching proxies) to make sound judgments of freshness and the lifespan of representations. The idempotence of a specific request operations (methods) permits representation reuse.

6. The presence of intermediaries is promoted. Filtering or redirection intermediaries may also use both the meta-data and the representations within requests or responses to augment, restrict, or modify exchanges in a manner transparent to both endpoints.

A unique characteristic of a RESTful service is that it prescribes a uniform interface. A RESTful service is exposed as the information resources upon which a fixed set of operations. In a REST solution, for each resource we have to define a representation. In most cases, XML or JSON format is used. Also, RESTful services are necessarily stateless, they do not store the conversational state across multiple requests from the same service user.

Web Application Description Language (WADL) [127] is a resource-centric description language. A WADL document is composed as a set of resource descriptions. In WADL, even a complicated business application is described as basic operations (*i.e.*, PUT, GET, POST, or DELETE) on the resources that comprise the state of an application. In addition, of WADL, there are other description languages emerging, such as Web Resource Description Language (WRDL) [135]. Many of the REST-based description or REST like description violates one or more core design principles. Some of the service providers use the web pages and the PDF files to describe resources and resource schema solely targeted to the human users.

The SOAP-based services and the RESTful services represent different paradigms to implement SOA. A SOAP-based service is centered on the operations to be executed by a service provider, and a RESTful service is focused on accessing resources. In the architecture evaluation of an SOA system, the evaluation team can question which approach would be more appropriate for each service. REST is an excellent option

for accessing static or nearly static resources. It is also useful for portable devices with limited bandwidth because the REST messages are less verbose than the SOAP messages.

Comparison between SOAP-based Services and RESTful Services

RESTful services are gaining popularity and are adopted by the leading service providers such as Google, Yahoo, Amazon and eBay [119]. According to an API listing site [100] 73% of the web services provided by the service providers are RESTful services. The total number of APIs in this site is 3456 as of January 22, 2014. Pautasso *et al.* [95] uses architectural principles and decisions as a comparison method to illustrate the conceptual and technological differences between the RESTful services and the SOAP-based services. On the principle level, both approaches have similar quantitative characteristics. On the conceptual level, less architectural decisions must be made when deciding for SOAP-based services, but more alternatives are available. On the technological level, the same number of decisions must be made, but few alternatives have to be considered when building RESTful services. Thus, the perceived simplicity of REST now can be understood from a quantitative perspective. Some key differences between SOAP-based services and RESTful services are as follows:

- **Idempotent operations.** HTTP methods such as GET are classified as idempotent. If GET is performed multiple times on the same resource, the results should be identical. Other methods (such as POST) are non-idempotent. If a POST is performed multiple times on the same resource, the side effects are undefined by the specification. In practice, most SOAP interactions are performed

via the POST HTTP method. Therefore, it is not possible to know whether a message will be idempotent without acquiring semantic knowledge of what the SOAP receiver will do with the message. However, in HTTP, by looking at the method name, idempotency of the method can be identified. The protocol itself defines whether the operation is idempotent without any relationship to the resource. This inability to rely upon idempotency presents a significant obstacle to intermediaries that wish to intelligently cache SOAP messages sent over HTTP.

- **Use of SOAP envelope.** The use of a separate SOAP envelope collides with the notion of separation of HTTP meta-data from data. The SOAP protocol binding contains the entire SOAP envelope in the body of the HTTP request. Therefore, in order to properly parse the request, a SOAP intermediary must examine the entire body of the HTTP request. An HTTP proxy can operate by examining the meta-data of the request.

- **SOAP-based services are less efficient for mobile clients.** With an increase of use in mobile devices, one can easily imagine that in the future, clients using mobile devices will generate a larger percentage of web service requests. Despite the fact that the condition of mobile computing has improved so much in recent years [88, 89], applying current web service communication models for mobile computing may result in unacceptable performance overheads. This potential problem comes from two factors: 1) Encoding and decoding of verbose XML-based SOAP messages consumes resources. Therefore, web service participants, particularly mobile clients, may suffer from degraded performance.

2) There is a performance and quality gap between wireless and wired communication. Hence the potential problems are due to the mobile environment's constraints such as limited processor speed, limited battery lifetime, and slow unreliable connection. Performance in mobile web services [27, 118] is an open research area. Several messaging optimization approaches have been introduced to address web service performance overhead for mobile clients. Using current web service communication models for mobile computing may result in unacceptable performance overheads. A typical web application requires the transmission of four to five times more bytes if implemented as a web service, compared to the same service implemented as a traditional dynamic program (*e.g.*, ASP/PHP) [118]. Since RESTful services come from the WWW architecture, its performance is similar to web pages, making the RESTful approach more suitable for mobile devices.

- **Interoperability.** SOAP-based services can interact with other SOAP-based services using remote process call over HTTP. However, REST resources can easily interact with other web resources (*e.g.*, web pages) using an HTTP library. In other words, any client or server application with HTTP support could easily invoke a RESTful service using HTTP methods.

- **Scalability.** In a typical REST interaction, requests for a representation of a resource (*e.g.*, HTML content in a web page) from a user agent (*e.g.*, a web browser) and responses from a server can be transferred over the web by passing through multiple caching proxies. If results are available in the intermediate proxies, the caching can offer quick response to the user agent without requesting to the ultimate server which hosts the requested resources. In this context, using

```
GET /country/capital?x0=Nepal HTTP/1.1
Host: 130.15.19.65:8080
Accept: application/xml
```

(a) RESTful Request

```
POST /country/capital HTTP/1.1
Host: 130.15.19.65:8080
Content-Type: application/soap+xml
<? xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Body>
        <GetCapital xmlns="http://org.myservice">
        <country>Nepal</country></GetCapital>
        </soapenv:Body>
</soapenv:Envelope>
```

(b)SOAP- based Request

Figure 2.4: Request in operation-oriented and resource-oriented service invocation

RESTful services can reduce network latency while facilitating service design and development is independent and scalable.

- **Two-level naming system.** The SOAP HTTP binding also suffers from a two-level naming system. Figure 2.4 shows an HTTP POST message to */country* resource. Inside the SOAP envelope, the SOAP body indicates that the *getCapital* method should be executed. The actual function is not known without examining the SOAP body. In this example, any intermediary that attempts to route a message would have to understand the entire */country* name-space. The granularity of the naming system is insufficient to allow an intermediary to intercept *getCapital* requests without looking at the body of the SOAP request.

However, doing so may be in violation of the SOAP specification. A request would be an HTTP request with no SOAP components, while the response would be a SOAP response embedded inside of an HTTP request. However, this leads to an asynchronous method of operation. Rather than replying with an HTTP response, a SOAP message would be in response. This solution allows regular HTTP proxies to cache the request using its normal mechanisms. However, this solution may cause problems for SOAP intermediaries. A SOAP intermediary would have to have two methods of interactions - a SOAP proxy and an HTTP proxy. This may lead to significant overhead for implementers of a SOAP aware proxy. If an active intermediary intends to rewrite the normal HTTP request, it must rewrite the request using the HTTP syntax.

2.2 Service Migration

Organizations face challenges to be more adaptable and transform to meet new customer demands with fewer resources and streamlining of its business activities. There is a growing move to introduce SOAs with their promise of cost-efficiency, agility, adaptability and legacy leverage. However, there are many aspects of transforming an organization to use SOA and many obstacles and issues that the organization has to address when introducing SOAs. Service-oriented architecture (SOA) migration is an architectural migration from any non-SOA system to a system that follows the service-oriented architecture principles, in order to achieve a new maintainable service oriented architecture implementation of the system. The major benefits of adopting service oriented architecture as a design framework is the ability to realize rapid and low-cost system development, to improve the overall system quality, and to better

enable integration with other systems. Lewis *et al.* [67] discuss a migration technique called the Service-Oriented Migration and Reuse Technique (SMART). The SMART technique helps organizations analyze legacy systems to decide whether their functionalities can be reasonably exposed as services in a service-oriented architecture. SOA is implemented as web services. Several studies in the literature have focused on the problem of migrating traditional legacy systems to web services.

Web services itself are evolving as new techniques are developed to provide faster and easier integration to the customers. One of such evolution is RESTful service from SOAP-based services. In this sub-section, we discuss migration of SOAP-based services to RESTful services and migration of web applications to RESTful services.

2.2.1 Migration of SOAP-based Services to RESTful Services

Kopecky *et al.* [65] present hRESTS as a solution for missing machine-readable web APIs of RESTful services. They argue that a microformat is the easiest way to enrich existing human-readable HTML documentation. They introduce a model for RESTful services, but with a focus on documentation and discovery. Alarcon *et al.* [5] introduce a meta-model for describing RESTful services as the basis for the Resource Linking Language. The authors identify links as first class citizens and focus on service documentation and composition. There are a number of work providing guidelines on designing RESTful services [66, 126] which provide some basis for our migration. Engelke *et al.* [38] present an industrial case study to migrate a transportation web service to a RESTful service. Laitkorpi *et al.* [66] describe an approach of abstracting application interfaces to REST-like services in three key steps: analyzing a legacy API, abstracting it to a canonical form with constraints in place, and generating

adapter code for the abstraction. Athanasopoulos *et al.* [12] provide a model-driven approach in identifying REST-like resources from legacy service descriptions. Using the information contained in the descriptions of the available functionality (in the form of WSDL or message schema specifications), the authors propose a way to model service operation signatures into a MOF (Meta-Object Facility) model, called Signature Model. All the approaches of identifying the resources, methods, and message conversion between the RESTful services and traditional web services are limited to identifying the resources. Several messaging optimization approaches have been introduced [88, 89] to address web service performance overhead for mobile clients. Current web service communication models in mobile computing may result in unacceptable performance overheads. A typical web application requires the transmission of four to five times more bytes if it is implemented as a web service compared to the same service implemented as a traditional dynamic program using ASP or PHP [118].

2.2.2 Migration of Web Applications to RESTful Services

A web application is accessed through a web browser running on a client's machine, whereas a web service is a system of software that allows different machines to interact with each other through a network. A web application is coded in a browser-supported language such as JavaScript (JS) and combined with a browser-rendered markup language, such as the hypertext markup language (HTML). Web applications are popular due to the ubiquity of web browsers and the possibility to update and maintain the web applications without distributing the clients. Web pages of web applications are defined in HTML and represented using the Document Object Model (DOM). All client side interactions are realized with modifying JS of the DOM. The presentation

of a web page is handled by Cascading Style Sheets (CSS). To understand a web application, a developer must be familiar with HTML, JS and CSS, and the interactions between them. Similar to the problems of early traditional applications, many web applications become legacy systems. Web applications are facing new challenges such as the integration of software provided by different organizations and the ability to create combined business scenarios. Web services provide system-to-system interaction and permit the implementation of business constraints using process control primitives to adapt the new challenges.

Sneed *et al.* [111] discuss a tool-supported method for legacy codes written in COBOL and wrapped behind an XML shell allowing individual functions within the programs to be offered as web services to any external user. Tatsubori *et al.* [116] present a framework named H2W, which can be used to construct web service wrappers for existing, multi-paged web applications. H2W's contribution is mainly its service extraction step. The authors propose a page-transition-based decomposition model and a page access abstraction model with context propagation. Tatsubori *et al.* [116] describe an approach to extract services by analyzing the client representation. But their approach does not extract resource relations. Almonaies *et al.* [6] present an approach to migrate web applications to a web service. The above mentioned approaches either require analyzing the server side codes or extract SOAP-based services without service relations.

2.3 Service Description Models

When publishing a service, the service providers hide the implementation details of a service. The service providers enable potential consumers to find and execute

services successfully, by advertising service description models. Service description models specify the capabilities of services, which usually includes the input/output parameters, the exceptions, and the functional and non-functional descriptions. For example, Web Services Description Language (WSDL) [34] uses Extensible Markup Language (XML) to describe the operations offered by a service, the input/output messages of operations, and the address to access a web service.

Table 2.2 classifies web service technologies into semantic and non-semantic categories. Furthermore, the two main orientations of web service styles, SOAP-based services and RESTful services, commonly use technologies which are service style specific. WSDL describes a single service, while BPEL4WS represents composite executable processes. However, these two technologies are placed into the same category because they are both parts of the same SOAP-based service. Figure 2.5 shows a fragment of WSDL. Web Services Description Language (WSDL) [34] is an XML-style language for describing a single service interface. The WSDL description provides information on how the service can be invoked, specifies the location of the service, the parameters it expects, and the parameters it returns. Most of the services found in the web are SOAP-based services.

Conventional web service description models specify the syntax of the data, but cannot describe the semantics of the data. Without the semantic description, the service consumers and the service integrator need additional background or additional documents to understand and access a service. The lack of semantic description makes automatic service composition difficult. To add semantics to web services, various semantic web service description models [34, 127, 135, 138] are proposed by researchers. Semantic web service description models are generally built on accepted

Table 2.2: Service description models

		Technology		Description
Web service description models	Non-semantic technologies	SOAP-based services	BPEL4WS [82]	Language for describing process composition; replaced by WS-BPEL
			WS-BPEL [136]	Language for describing process composition
			WSDL [34]	Language for a single web service description
		RESTful service	WADL [127]	Language for describing HTTP-based service
			POWDER [92]	Protocol for describing a group of resources
			RIDDL [103]	Language for web service description
			hRESTS [65]	HTML micro-format for RESTful web services
	Semantic technologies	WSML [137]		Language for specification of ontologies and web services
		OWL-S [91]		Language for describing semantic-based services
		WSMO [138]		Framework and language for describing semantic web services
		SOAP-based services	SAWSDL [58]	Semantic annotation for WSDL
			WSDL-S [115]	Language for describing semantic web services
		RESTful services	SA-REST [61]	Format for adding annotation to REST API description in HTML or XHTML
			POWDER-S [1]	Semantic POWDER
			EXPRESS [8]	Approach for expressing semantic services using domain ontologies

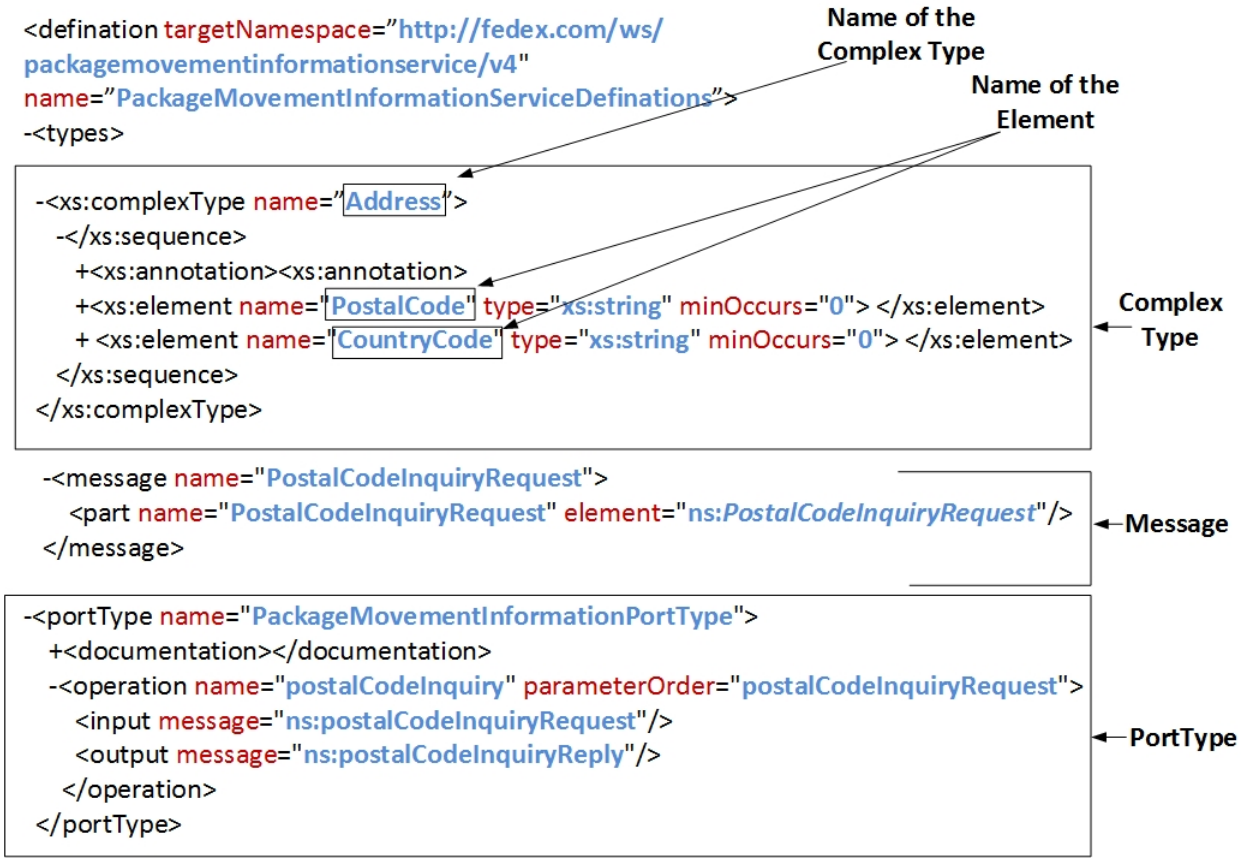


Figure 2.5: Annotated WSDL document

standards to exchange semantic data. Semantic data decrease the difficulty for service consumers and machines to understand the service description. Semantic Annotations for WSDL (SAWSDL) [58] and Web Service Semantics (WSDL-S) [115] define mechanisms for adding semantic annotations to WSDL components. Specifically, they provide a means of referencing semantic models from within the WSDL. In addition, several semantic web technologies, such as Web Ontology Language for Services (OWL-S), and DARPA Agent Markup Language for Services (DAML-S), are not restricted to a specific web service style; in Table 2.2, they are therefore included directly in a semantic technology category. RESTful services communicate

over standardized interfaces such as HTTP, by exchanging representations of the resources. RESTful services conform to the principles of the REST architecture style. WADL [127], POWDER [1], and hRESTS [65] are approaches for describing RESTful services. RESTful specific semantic technologies include Semantic annotations for REST (SA-REST) [61], Semantic POWDER (POWDER-S) [1], and EXPRESS [8]. SA-REST provides a means of adding annotations to REST API descriptions in HTML or XHTML. POWDER-S provides formal semantics for POWDER. Specifically, POWDER-S aims to make POWDER data available for semantic web tools by representing POWDER data as an OWL ontology. The EXPRESS approach specifies RESTful semantic services using domain ontologies. Subsequently, RESTful interfaces are created from ontology classes, instances and relations. However, most of the services are described in HTML web pages and PDF files.

2.4 Service Discovery

In this section, we describe the different approaches of service discovery. To consume a service, the developers should be able to find and invoke the service. Service providers advertise services in the service registry and enable service consumers to search for services from the registry. A service registry is a bridge between service providers and service consumers and enables service consumers to access a wide range of web services that match specific searching criteria. The ability to discover a service provider if there is a service with the properties described in the request is called service discovery.

Search engines [15, 49, 134] have become a new source for searching web services. Service engines can help us to find the service description documents, such as WSDL. Table 2.3 describes three service discovery (*i.e.*, information retrieval, semantic and

Table 2.3: Service discovery approaches

Service discovery approach	**Example approaches**	**References**
Information retrieval approach	Keyword-based, classification, Link ranking	[45, 46, 55, 105, 128]
Semantic-based approach	OWL-S, Micro-formats, hREST	[65, 81, 84]
Contextual data based approach	OWL to model Context	[13, 16, 19]

contextual data) approaches along with the example approaches. We can use these service description documents to access services over a network. Moreover, some service search engines, such as Seekda [134] and Woogle [134] are particularly designed to retrieve web services. Service consumers can use keywords or some specific query languages to find services using these search engines. After service registries or service search engines collect service descriptions, the service discovery system needs to find proper services based on requests from potential service consumers. In this section, we classify the service discovery mechanisms into two categories: information retrieval approaches, semantic matching approaches and contextual data approaches.

2.4.1 Information Retrieval Approaches

Document matching and classification are the typical problem in the field of Information Retrieval (IR). Many researchers use IR techniques to find services. In this sub-section, we describe the three different approaches of information retrieval approaches of service mechanism mechanisms.

Keyword based Matching

Keyword matching is the most common form of text search in the field of IR. It retrieves the document by matching the keywords provided by the user and the keywords extracted from the text. UDDI [123] uses tModel to represent an abstract interface which helps to enhance service matching process. Sajjanhar *et al.* [105] propose a Singular Value Decomposition (SVD) based algorithm to locate matched services for a given service. Their approach considers the textual description and cannot reveal the semantic relationship between web services. Wang *et al.* [128] discover similar web services based on structure matching of data types in WSDL. The drawback is that simple structural matching may be invalid when two web service operations have many similar substructures on data types. Given a search query (*i.e.*, keywords), the service discovery systems search the textual descriptions and return a candidate answer set. Users can select the appropriate one from the answer set. However, simple keyword matching cannot use the semantics represented by structured documents. Therefore, keyword matching approaches cannot distinguish words that are spelled the same way but have different meanings. To use the semantics of documents, WSXplorer [53] analyzes the structure of the input/output parameters of the operations provided by a service and converts the input/output structured data into trees.

Classification

There are two main approaches for the automatic classification of web services, namely, the heuristic [90] and non-heuristic [20, 55]. Two different strategies are proposed by Hess *et al.* [55] uses the information contained in non-semantic service

descriptions to select a category in which the service fits the best. The strategy uses Natural Language Processing (NLP), machine learning and text classification techniques. The second strategy uses the same information contained in service descriptions to dynamically create the categories in which the service should be classified, using clustering techniques. In both strategies the classification process is based on the extraction of relevant words from service descriptions, the construction of term vectors with those relevant words and the usage of classification mechanisms (*e.g.*, nave bayes) to perform the vector classification. From our point of view, this approach has the following two main drawbacks:

1. Hypothesis of finding relevant and meaningful words in service descriptions is a very optimistic starting point.

2. Clustering implies that the created categories do not have meaningful names and the taxonomy changes over time.

These problems can be an issue if the classification information is intended to be used for service discovery, since users could neither select services by category, nor get properly acquainted with the taxonomy, as the category structure may change frequently. Dong *et al.* [32] present a search engine called Woogle which is able to make similarity search between web services. The similarity search combines multiple techniques (*i.e.*, similarity primitives) to compare web services and their operations. The primitives are applied to different levels of a web service description: input and output parameter names, operation description and web service description. Dong *et al.* [32] introduce algorithms to cluster terms in the descriptions into concepts. They use information retrieval metrics (such as term frequency-by-inverse document

frequency) to measure the similarity between concepts. Peng *et al.* [96] present a method to classify and retrieve web services in a repository using formal concept analysis. The authors propose to build a concept lattice starting from a context where objects are web services in a UDDI repository and attributes represent the operations in these services. In this approach, optimizations are made to reduce the size of the operation set used for building the lattice. Similarity search techniques are used to compare operation descriptions and input/output message data types, extracted from WSDL files. Similar operations are merged and one operation among these ones is put in the context table. References to the other similar (merged) operations are stored in an external data structure to easily find them during web service retrieval. Peng *et al.* [96] propose an algorithm for retrieving web services. The algorithm parses the lattice in a breadth-first manner from top to bottom to find the concepts (*i.e.*, services and operations) that match queries defined by users. These queries are first abstracted as a formal sequence of operations to be handled by the algorithm. Arabshian *et al.* [11] use service classification ontology to classify services into different classes. When a user searches for services, the service classification ontology identifies the domain of the query, and guides users to provide more detailed data to trace the service classification ontology from a high-level class to concrete services. There are often problems involved in service categorization which causes difficulties for the creation, validation, and use of classifications in real-world environments. More specifically, we list three major problems as follows:

1. Taxonomies can be extremely large, comprising thousands of categories (*e.g.* UNSPSC [124] about 20,000 classes, NAICS [87] about 2,300 classes).

2. The number of services in a service repository can grow quite large, making it

virtually impossible to validate the information published along with a service.

3. Knowledge of taxonomy is required to place a service under a proper category. Few publishers have the sufficient knowledge.

Link Ranking

If there is more than one service matching the required functionality, it is necessary to rank the matching services and select the best one. Service rank is a quantitative metric to show the importance of a service within a web service network. Service ranking helps to select services [45]. Similar to PageRank [18], which uses the links among different web pages to evaluate the importance of web pages, Gekas *et al.* [45, 46] propose an approach to rank services using dynamic data flows among web service operations. Gekas *et al.* methodology uses the search space that consists of all the potential web service operations to be a part of a workflow. Gekas *et al.* approach automatically collects the extracted information about the connectivity structure of a semantic web service repository (*i.e.*, the registry) and uses the extracted information in order to guide the search process.

2.4.2 Semantic Matching Approaches

A limitation of traditional information retrieval approaches is their lack of semantic matching [93]. Service providers and service consumers have very different perspectives and knowledge about the same service. It is unrealistic to expect that service providers and service consumers would give the same description of services. Furthermore, the service providers would not provide exactly the services that a service

consumer needs. For example, a service provider may describe a service as a financial news provider, while the service consumer might need a service to offer updated stock information. We cannot match a service request with a service description by using traditional information retrieval approaches. The notion of semantic web services [81] goes one step closer to the interoperability of autonomously developed and deployed web services, where a software agent or an application can dynamically find and bind services without having a prior hard-wired knowledge about how to discover and invoke them.

OWL-S [91] is a specific OWL ontology designed to provide a framework for semantically describing such services from several perspectives (*e.g.*, discovery, invocation, and composition). During the development of a service, the abstract procedural concepts provided by the OWL-S ontology can be used along with the domain specific OWL ontologies which provide the terms, concepts, and relationships used to describe various service properties (*i.e.*, inputs, outputs, preconditions, and effects). Microformats [84] are a set of simple, open data formats built upon existing and widely adopted standards. There are microformats for contact information, geographic coordinates, calendar events, and ratings. REST uses microformat approach to introduce semantics. hREST [65](HTML for RESTful services) is a microformat for machine-readable descriptions of web APIs, backed by a simple service model. hREST provides constructs to markup operations and data elements in an API description.

The problem with the semantic based matching approaches is that it is not widely used, and it is difficult to convince service providers to use one particular vocabulary or ontology. Thus, even if it is adopted, there are different independent vocabulary and ontology.

2.4.3 Context based Approaches

Balke *et al.* [13] propose an algorithm to select a web service based on user's preferences. The algorithm starts with a general query. If there are too many results, it expands the service query using a user's preferences. Firstly, the algorithm expands the query with hard constraints extracted from user's preferences. If there are still too many results, it extends the query with soft constraints (*e.g.*, user supplied context information) and searches for web services again. By adding constraints step by step, the algorithm narrows the number of service searching results down to a small value. Blake *et al.* [16] use an agent to detect the contextual data from executing applications and behaviors of human users, such as browsing the Internet. Based on the contextual data, the agent generates a query action to search for available web services. Their agents recommend the services by matching the similarity among the inputs. Broens *et al.* [19] use ontologies to model contextual information and service descriptions. Service requesters, service providers and context providers share the common knowledge defined in ontologies. Four different service properties are defined in their service matching algorithm: service types, outputs, inputs, and contextual attributes. The service type is a reference in an ontology or a classification of services. Outputs and inputs are concepts defined in ontologies to specify the meaning of inputs and outputs. Contextual attributes represent the contextual information obtained from service requesters and service providers. The algorithm use service types, outputs and inputs to search for services, and then use contextual attributes to sort services.

Table 2.4 provides a comparison of different service discovery approaches. We compare different service discovery approaches in the context of service description language, context data and support for user's to search services. Use of context is

Table 2.4: Comparison between different service discovery methods

Service discovery method	**Description language**	**Service description language**	**Assist users in query formulation**	**Context data**
UDDI registry [123]	Searching by categories (classified by service providers) and keywords matching	UDDI and WSDL	No	No
Woogle [134]	Cluster services according to parameter names of inputs, outputs and operation names	WSDL	No	No
Link Ranking [45, 46]	Ranking servers using the dynamic data flow among web service operations	OWL-S or DAML-S	No	No
WSXplorer [53]	Matching the structures and keywords of input and output parameters	WSDL	No	No
Broens *et. al* [19]	Use service type, output and inputs to search for services; use contextual data to sort services.	Ontologies defined on top of WSDL	No	No
Benatallah's model [14]	Treat the service matching as the best covering problem in description logic area.	DAML-S	No	Yes

increasing in the service discovery mechanism. Similarly, as shown in the Table 2.4 none of the approach helps a user to formulate the query.

2.5 Service Selection

Service selection is a very complex and challenging task, especially if it takes a variety of different non-functional properties into account. Al-Masri *et al.* [4] reports that there is more than 130 percentage growth in the number of published web services. Similar observation can be made by reviewing the statistics from web service search engines such as Seekda [107]. In particular, Programmable web directory [100] indicates an exponential increase in the number of web services over the last three years. The rapid growth in the number of services increased the importance of the service selection task due to the presence of low quality services. In the state of the art approaches for service selection in order to filter out low quality candidates during the selection process, non-functional aspects are exploited as the key decision making criteria. As a result, quality of service (QoS) is a significant concept since QoS properties describe non-functional aspects of web services and evaluate their conformance degree. The fundamental challenges of service selection are (1) specifying service requirements, (2) evaluation of the service offerings, and (3) aggregating the evaluation results into a comparable unit. The customer's requirements and the service offerings have both functional and complex non-functional aspects, which need to be expressed and for evaluation matched against each other (which is often only possible in some partial way). In this section, we discuss service selection based on non-functional quality of service.

Another mechanism used in web service selection relies on trust and reputation. A service consumer chooses a service that is trusted or a service with a high reputation. There are some other aspects that may impact a service selection, such as payment methods, penalty rates, and location. No matter whether web services are used for

business-to-business interactions or business-to-consumer interactions, web service selection may need to be customized according to the users' different constraints and preferences. The web service selections for flight booking and hotel reservation have to be personalized according to a user's constraints and preferences. Most approaches proposed in the literature about personalized selection concentrate on how to rank web services according to users' preferences on various QoS metrics. The existing approaches in service selection fall into the following categories.

2.5.1 Policy-based Service Selection

Policy based service selection approaches [59, 75] specify the non-functional requirements by coding these in a QoS policy model or policy language. The QoS policy model in [75] is designed as a textual document. Lui's [75] approach offers two types of non-functional properties: generic and domain specific ones. The domain specific properties are extensions of the generic ones for different kinds of services. The content of the policy model represents the service requester's non-functional constraints and preferences. Lui's approach defines two universal evaluation functions (one is used for the case where a lower value benefits the requester and the other one are for the opposite case) to evaluate the service's non-functional properties against the policy model. The relations between the non-functional criteria are expressed in a matrix, which is also used for their aggregation. One of the most crucial disadvantages of this approach is the human involvement. Firstly, it is difficult to formalize all the non-functional criteria in order to allow computation of the overall score. Secondly, all the non-functional properties have to be presented as numbers or be converted into that format. Thirdly, while it captures how to express requirements, there is no mention

of where and how values of the properties' are stored and expressed. Fourthly, the matrix aggregation function is difficult to understand for users and does not show the relation of user's preferences at all. Finally, the final ranking scores do not reflect the satisfaction level that can be expected from the service as the overall range of values is not specified.

A similar approach is introduced by Janicke *et al.* [59], the improvement being the formalization of the non-functional properties into a conditional policy language. The other differences are that the service properties are dynamically detected by hardware sensors monitoring whether the selected service breaks the requestor's requirements. However, this feature limits the number of properties for practical reasons (it is only possible to monitor a small range of properties), and of course is of limited value to decisions on service selection before execution as the monitoring is performed during execution. This technique could however be very useful combined with a selection strategy that uses service execution history.

2.5.2 Trust and Reputation-based Service Selection

Several trust and reputation-based approaches have been proposed for web service selection. Most of these approaches depend on a central QoS registry to collect and store feedbacks from consumers. The general idea of these methods is that consumers report the data acquired from executing a web service (*e.g.*, execution time, response time) and their ratings on other QoS metrics. According to the QoS information and a consumer's profile that shows the consumer's preference over different QoS metrics (*i.e.*, how these QoS metrics are important to a consumer), the QoS registry calculates an overall rating for each web service that matches the consumer's search request.

Then, the consumer selects the web service with the highest rating. Although these approaches share the same idea, they are different in their focuses and calculation algorithms. Maximilien *et al.* [79] put a lot of effort on building a QoS ontology, the basis for service providers to advertise their services and for consumers to express their preferences and provide ratings. Liu *et al.* [75] propose an algorithm about how to combine different QoS metrics to get a fair overall rating for a web service. Manikrao *et al.* [77] use the collaborative filtering technology in their web service selection method. They use some dedicated QoS registries to collect QoS feedbacks from consumers. Although these QoS registries are organized in a P2P way, they are based on a specially designed P-Grid structure. Each registry is responsible for managing reputation for a part of service providers. An algorithm is introduced to detect and deal with dishonest feedbacks by comparing the QoS data from dedicated monitoring agents with the data from consumers to filter out dishonest feedbacks. This approach is much more complicated than those centralized trust and reputation methods and involves a lot of communication and calculation because of the use of the complicated P-Grid structure.

2.6 Service Composition

Service composition lets the users create applications on top of the native service description, discovery, and communication capabilities of service oriented computing. Such applications are rapidly deployable and offer reuse possibilities for the developers and seamless access to a variety of complex services to users. Service composition can be either performed by composing elementary or composite services. Composite services in turn are recursively defined as an aggregation of elementary and composite

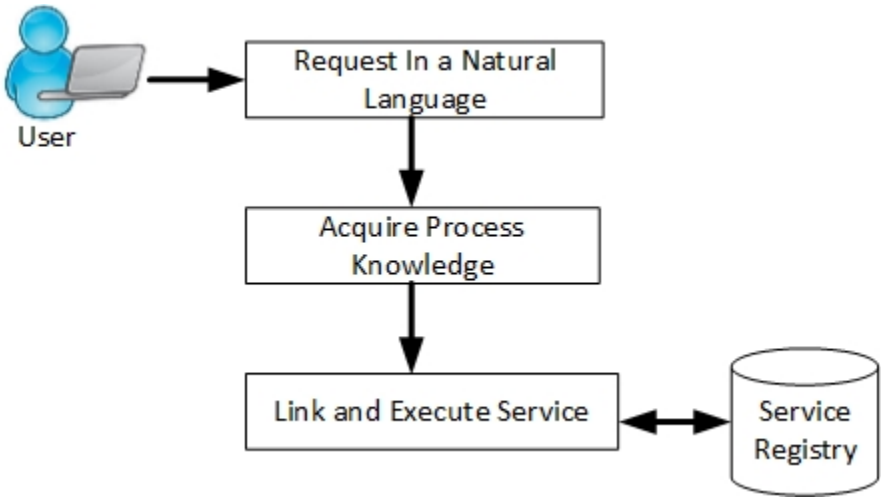


Figure 2.6: Process of service composition

services. When composing web services, the business logic is implemented by several services. It is analogous to workflow management, where the application logic is realized by composing autonomous applications. This allows the definition of increasingly complex applications by progressively aggregating components at higher levels of abstraction. A client invoking a composite service can itself be exposed as a web service. Some of the service composition solutions [142, 143] identify the need of QoS attributes to get the optimum solution.

Figure 2.6 shows the process of service composition. In most of service composition approaches found in the literature are based on formal language. If the service customer is a non-technical user, she/he specifies the goal in the natural language. This goal needs to be broken down into different tasks which in turn is mapped to the services. There may be more than one service; the service selection may depend on the user preferences, quality of service and historical information. Service descriptions specify the capabilities of services, which usually include inputs/outputs, exceptions,

functional and non-functional description. Service selection needs to consider different metrics to select the service (such as trust of the service, non-functional properties and input and output compatibility). Service selection may require the service customers' input. Once all the related services are selected, the flow of data between the services can be either automatically generated using different AI planning algorithms or it can be defined by the user. There are different service composition frameworks.

2.6.1 Acquire Process Knowledge

Process knowledge provides the required data to transform business process requirements into business processes and compose services. In model-driven service composition, process knowledge is primarily provided by experienced business analysts who have extensive process knowledge. In goal-driven service composition, most approaches are built on semantic web services which contain process knowledge (*e.g.*, pre-conditions and post-conditions) to reason business processes. However, there is no well-accepted semantic service description model. For example, the two most popular semantic service description models, OWL-S and Web Service Modeling Ontology (WSMO) [138], are not recommended by W3C. In practice, WSDL is the default service description language. WSDL does not have semantic description and does not have the ability to describe process knowledge. It is challenging for novice business analysts and non-professional users to identify a complete set of services to orchestrate a well-defined business process due to the lack of process knowledge. Therefore, it is necessary for service composition systems, especially for the systems which are not built based on semantic web services, to have the required knowledge to compose services. There are two major research areas involved in capturing process knowledge,

namely process mining and information extraction.

Process Mining

Process mining is a technique to extract business process information from event logs recorded by information systems. Agrawal *et al.* [2] present an approach to construct process models from the log. The approach can generate a process model with the control flow of the business process from the logs of unstructured executions of a process. Francescomarino *et al.* [31] trace the web system executions and analyze the application Graphical User Interface (*i.e.*, the forms and their fields on the client web page) during the execution to infer processes. Liang *et al.* [69] provide an approach to mine service association rules from service transactional data. Aalst *et al.* [125] use Petri nets to model processes and discuss the class of processes which can be discovered from the logs, then they propose a mining algorithm to discover business processes. However, the business processes mined by the aforementioned approaches are formally defined and are not used to handle daily activities. Moreover, the event logs for daily activities are distributed on many servers. It is hard to collect the event logs from different places, especially when it involves personal data.

Information Extraction

Information extraction systems transform unstructured documents or semi-structured documents into structured data such as, a relational database. Cimiano et al [29] develop a framework to learn ontologies from text documents. Chang *et al.* [26] summarize and compare the existing web information extraction systems which extract information from semi-structured documents (*e.g.*, HTML documents). Chang *et*

al. compare the major web data extraction approaches based on the task of the IE systems (*i.e.*, the types of input documents and the extraction targets), the automation degree, and the techniques used (*e.g.*, extraction rules, approach to assemble the extracted values). Yoshida *et al.* [144] provide an unsupervised learning method to extract ontologies from the tables shown the web pages. However, the aforementioned approaches are designed to extract general information from documents instead of the process knowledge, which makes us difficult to use the extracted information to generate processes and compose services. Liu *et al.* [73, 74] develop a process-based search engine to search process knowledge from the web. The results of the search engine are text description. Their process-based search engine can retrieve the processes explicitly published on the web, such as E-How [36]. However, their approach can only extract the process knowledge from the web pages with explicit process information. Meanwhile, the output of Liu's work is the unstructured text description about a process. It is difficult to be further processed by machines. Hoxha *et al.* [56] provide an approach to extract semantic descriptions of processes in the web. Hoxha's approach fills the forms on websites, and recovers the process following the submission buttons step by step dynamically. However, to automate Hoxha's approach, it requires automatic approaches to fill online forms, which are difficult to achieve nowadays and may impact the execution of the online applications.

2.6.2 Link and Execute Services

As we have pointed out above, the term composition denotes not only the result of composing web services into a new, larger, course of action. It is also used to describe the process of composing. Different strategies for composition can be identified.

Schahram *et al.* [33] identify the following two categories of composition approaches: static vs. dynamic service composition and automated vs. manual service composition. BPEL and WS-CDL are aimed at manual service composition, whereas OWL-S is aimed at automatic composition. We give a short introduction to the approaches below.

Static vs. Dynamic Service Composition

The difference between static and dynamic composition deals with the point of time at which a concrete web service is integrated into the specification of a composition. Using static composition, the concrete services are determined and integrated into the specification at design-time. Web services are an active area where new services are added and existing services change or disappear. The constant change in services implicates that every change of an already integrated service has to be taken into account in the specification in the static composition approach. With dynamic composition on the other hand, there is only a specification of the type of services given at design-time. The concrete service is then integrated at run-time. Thereby it is possible that the concrete service has to be discovered first or that it is already known at run-time. The composition languages, such as BPEL and WS-CDL, are provided for a static composition approach. Xiao *et al.* [139] provides an approach that hides the complexity of SOA standards and tools from users and automatically composes services to help a user fulfill their daily activities. Xiao *et al.* [139] propose a tag-based service description to allow users to understand the functionality of a service and add their own descriptive tags. Using Xiao *et al.* approach, a user only needs to specify the goal of their activities using keywords.

Automated vs. Manual Service Composition

Alonso *et al.* [7] describe a development environment for manually composing web services into a larger course of action. A graphical user interface is provided where the designer can drag and drop web services and compose them in a graph like way into a control flow. The goal of automated service composition is to automate search, selection, and composition of web services. The service composition is meant as the specification of the sequences such as an ontology-based composition approach is examined [91]. Therefore, it is necessary for the respective web service to have a standardized and computer-interpretable description of the properties and capabilities. As WSDL does not provide any semantic description for a web service, the web ontology language for web services (OWL-S) is intended to provide the semantic description. The result of automatic composition [46, 140] can be expressed using a composition language, such as BPEL or WS-CDL. Thus, one can pertain compatibility with established (BPEL) or emerging (WS-CDL) standards for the description of composite web services.

2.6.3 Service Composition Frameworks

We present a survey of existing service composition frameworks. We outline the types of service component those each architecture supports; the methodology that it employs for service composition, and analyze the ability of each system to recover from composition request failures and run-time service failures. We briefly describe different service composition frameworks and table 2.5 summarizes different approaches.

Table 2.5: Comparing different service composition frameworks

	eFlow	**Ninja**	**ICrafter**	**SAHARA**	**Anamika**	**Argos**	**Composer**	**TCF**
Distributed Model	No	No	No	Yes	Yes	-	No	No
Supports Context	No	No	No	No	No	Yes	Yes	No
Composition Strategy	DC	DC	DC	DC	DC	DC	DC	DC
Component Technology	XML	XML, SDL	XML	XML	DAML-S	RDF	DAML-S	DAML-S, UPnP
Composition method	Graph	Graph	Graph	DAML-S	DAML-S	BPEL4WS	DAML-S	DAML-S
Composition failure recovery	No	No	No	Partially	Partially	-	-	Partially
Execution Failure Recovery	Yes	Yes	Yes	Yes	Yes	-	-	Yes

- means Information not found

DC means Dynamic Composition

Composer [109] is a semi-automatic service composition framework. The basic functionality of Composer is to let the users invoke web services annotated with DAML-S. A user is provided with a list of services registered to the system and can execute an individual web service by entering the values for the input parameters. The DAML-S services are executed using the WSDL grounding information. Composer is the first system to combine the DAML-S semantic service descriptions with actual invocations of the WSDL descriptions allowing us to execute the composed services on the web. Using the Composer, one can generate a workflow of web services. The composition is done in a semi-automatic fashion where Composer presents the available choices at each step and a human user makes the selection among available services. Compositions generated by the user can be saved as a new service which can be further used in other compositions.

eFlow [23] is a system for on-line, adaptive composition of e-services. eFlow models compose services as a graph defining the order of their execution. Graphs contain three types of nodes: service nodes, decision nodes and event nodes. Decision nodes carry flow control rules. Event nodes enable service processes to send and receive information about suspension, completion, and failure of services. eFlow performs composition in a centralized way. It uses a service broker to discover a service, which can fulfill the requests specified in the service node definition. eFlow is based on Java and is compliant with workflow and Internet standards, such as XML and the Workflow Management Coalition Interface, targeting fixed infrastructure type services.

Chandrasekaran *et al.* developed Ninja [25], a system for automated composition of existing XML-described services through heterogeneous devices and networks,

given Quality of Service metrics. Ninja is based on a cluster computing platform, which utilizes redundant control paths to enable fast fault-recovery. The central element of Ninja is Automatic Path Creation (APC), a component that identifies a set of services and the corresponding network connectors for devising and executing a composite service. To devise a composite service, APC creates a logical path by searching over a graph of the service space using the shortest path strategy. It then creates a physical path to locate service instances, which are used to instantiate, execute and monitor a composite service. Ninja places and locates services and dynamically adapt to resource availability to achieve better resource utilization.

ICrafter [98] is a service framework for a class of ubiquitous computing environments known as interactive workspaces. An interactive workspace is a physically co-located, technology-rich space consisting of interconnected computers (such as desktops, laptops, and handhelds), utility devices (such as scanners, and printers) and I/O devices (such as table-top displays, microphones, and speakers). The objective of ICrafter is to let users flexibly interact with services in their environment using a variety of modalities and input devices. ICrafter provides an infrastructure for UI selection, generation, and adaptation to offload services and user input devices. It is designed with the aim of automatically creating UIs for composite services. The ICrafter architecture utilizes a central composition model, which supports both off-line composition using service templates, and on-line composition. Services are described using an XML-based Service Description Language (SDL).

SAHARA [101] is an architecture for creation, deployment, and management of services. SAHARA enables composition across independent service providers. It employs a layered reference model, where services range from providing basic network

reachability and creating overlay networks, to instances of application building blocks, requiring processing and storage. Services are made available over the Internet and are composed by service level paths. SAHARA supports both centralized and distributed composition models. It embodies different mechanisms: measurement-based adaptation, utility-based resource allocation, trust management, service verification, and policy management. Finally, it allows for heterogeneous service composition across different service providers.

Masuoka *et al.* [78] develop the Task Computing Framework (TCF) [78] to allow users to compose and execute complex tasks in ubiquitous environments. Their architecture provides a Task Computing Client to discover, create, manage and manipulate services. TCF allows for both on-line and off-line composition. It embodies a centralized composition model to assemble semantically enriched web services, described in DAML-S, as well as plain UPnP services.

Chakraborty *et al.* present Anamika [24], a distributed, decentralized architecture for dynamic service composition in pervasive environments, based on a peer-to-peer model. Services are described using DAML-S and incorporate information about inputs, outputs, functional classification, as well as platform specific information, such as processor type. This is used by the composition broker to reason the possible service compositions. The Anamika architecture is tolerant to faults arising from service and network unavailability. The peer-to-peer model allows any devices to act as a broker facilitating service composition, and making the design immune to a single point of failure.

Argos [9] focuses on the automatic composition of web services workflows. Argos is based on the precise logical descriptions to the inputs and outputs of each web service

based on the ontology of the application domain. Argos systems do not assume that the inputs and outputs are simple types, such as strings or numbers, but that the services consist of complex data types. Reasoning the relational descriptions allows the Argos planner to automatically compose a workflow in response to a user data request and input/output compatibility.

2.7 Summary

This chapter describes the service oriented architecture and compares different types of services. We outline the background and the related work of different stages of service composition. We summarize techniques for service discovery and service selection approaches. Increasing growth in web services makes finding a specific service difficult. Based on the literature review, we find that service composition needs to address some major research issues to bring a user into the service composition process. Users have their own requirements and preferences on the composition results. Thus, making the process more compatible to non-technical user is a critical issue. We address some of the problems in service composition in this thesis.

Chapter 3

Identification of RESTful Services from Heteregeneous Services

Web services provide rich functionality for organizations and promote interoperable interactions over a network. Web services are mainly realized in two ways: 1) SOAP-based services and 2) RESTful services. For the service providers, the RESTful services can improve system flexibility, scalability, and performance as compared to the SOAP-based web services. The RESTful services are attractive to the end users because of low resource consumption (*i.e.*, battery, processor speed, and memory). Additionally, the RESTful services do not involve complex standards and diverse operations. Similarly, designing a reusable web application component is often neglected by the web developers due to the pressure for the time-to-market. Hence, in this chapter, we describe an approach to identify resources (*i.e.*, RESTful services) from SOAP-based services. Our approach also identifies tasks from web applications and presents these tasks as RESTful services. Heterogeneous web service description languages restrict a search engine to locate suitable web services to fulfill the users' goal. So, we present a unified schema to represent RESTful services.

3.1 Motivation

Different services use different service description languages. WSDL [34] is the standard description language for SOAP-based services. RESTful services have different description languages, such as WSDL [34], WADL [127], and Relational Link Language (RELL) [5]. Some of the service descriptions for RESTful services are written in HTML requiring human involvement to understand and implement the services. It is challenging for the end users to understand and parse the different service description languages.

SOAP-based services involve sophisticated tools to invoke them. Moreover, SOAP-based services are heavy-weight services and the encoding and decoding of XML-based SOAP messages consumes computation resources (*i.e.*, battery, processor speed, and memory). RESTful services use HTTP clients such as web browsers. RESTful services describe a set of HTTP methods (GET, POST, DELETE, and PUT) for each resource. Each method has well defined semantics. RPC-based services also use HTTP clients, but all methods are tunneled though the POST operation. A web application uses forms and hyperlinks to perform a task. Each type of web service has its own protocol. There is no standard way for communication and interaction among different types of web services. Because of the benefits of RESTful services explained in Chapter 2 (Section 2.1), we represent and invoke all services as RESTful services.

To bridge the gap between the diverse services and their descriptions, we propose an approach to identify RESTful services from SOAP-based services and web applications. We also describe a unified model to represent the RESTful services.

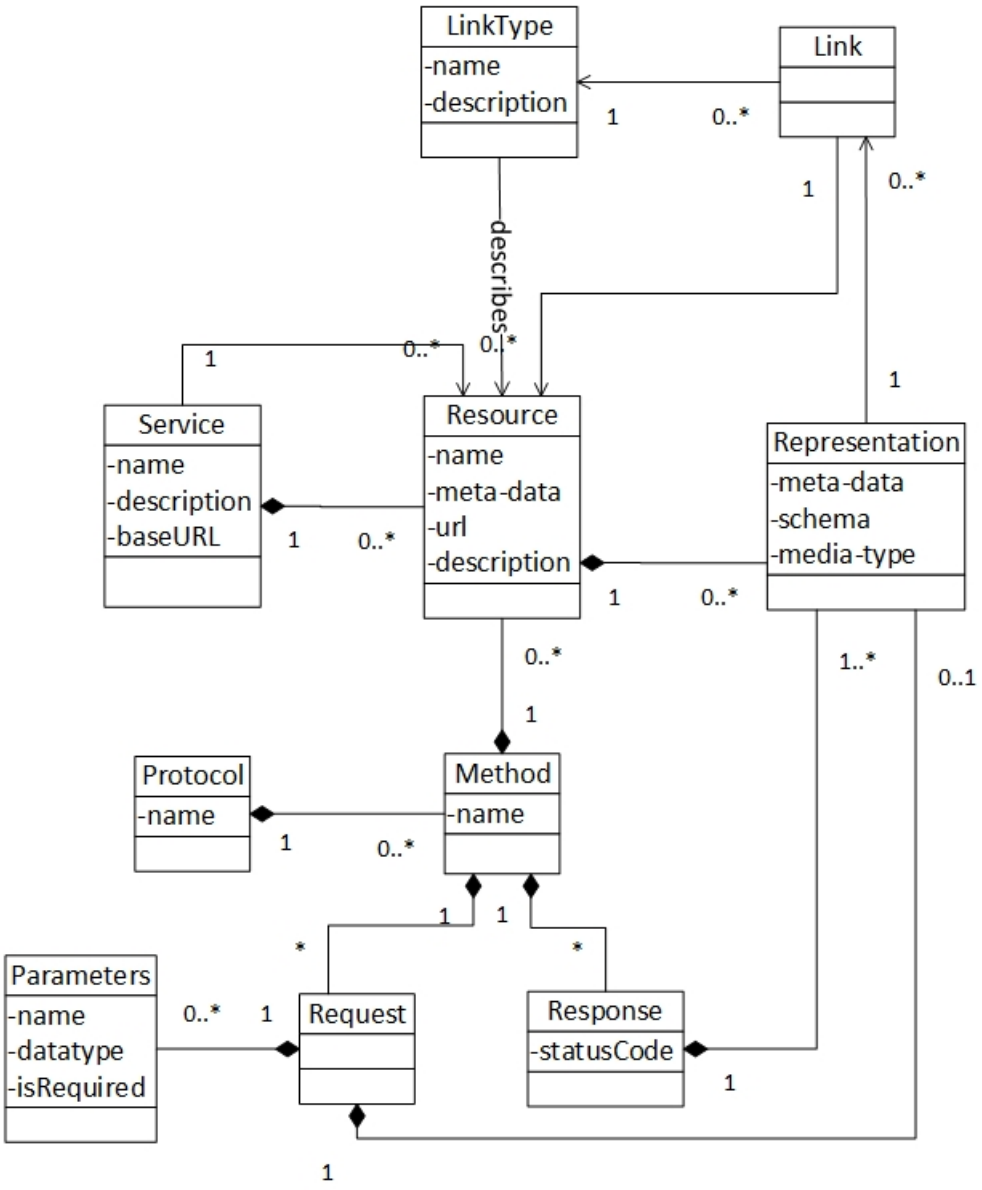


Figure 3.1: Schema to represent RESTful services

3.2 Modeling RESTful Services

Figure 3.1 represents a schema for the RESTful services. A service is a collection of resources by a particular service provider. A service has a base Universal Resource

Identifier (URI). A resource represents an entity whose state is exposed and can be changed via accessing the URI. We represent services in REST style. Resources are accessed using methods defined by a protocol. A method contains a request and a response message. For example, in the HTTP protocol [57], the methods can be GET, POST, PUT, and DELETE. A GET request for a URI notifies the service provider to retrieve the data. A PUT request indicates that a service provider should update the old data with the data sent in the request. A DELETE request indicates that a service provider should delete the data associated with the service requested. A POST request is used to create a new resource. A method can have at least one request and several optional response messages. The request includes a number of additional parameters to be sent to the resource. A response message contains a representation that returns the state of the resource. The response uses different status codes to represent correct and erroneous responses (*e.g.*, incorrect parameters, and server errors). The protocol of the method specifies the different types of status codes that define how response should be interpreted. For example, HTTP status code 200 represents that the request has succeeded. The information returned with the response is dependent on the method used in the request. For example, when the GET method is invoked an entity corresponding to the requested resource is sent in response.

```
<Service>
<Name> Flight Search </Name>
<Description>Search and bok the flight</Description>
<Provider>Flight Network <Provider>
<BaseURL>http://http://www.flightnetwork.com</BaseURL>
<Resource>
    <Name> Flight Search </Name>
    <Meta-data>
        <name= "Search"/>
        <name= "Flight"/>
        <name= "tickets"/>
    <Meta-data>
    <URL> http://www.flightnetwork.com/flights/search</URL>
    <Protocol type="HTTP" ref="http://tools.ietf.org/html/
rfc2626">
    <Method name="POST">
    <Request>
        <Parameter>
            <name="Category" datatype="String" />
            <name="From" datatype="String" />
            <name="To" datatype="String" />
            <name="From" datatype="String" />
            <name="Departure Date" datatype="Date" />
            <name="Return Date" datatype="Date" />
            <name="Adult(s)" datatype="Integer" />
            <name="Child(2-11)" datatype="Integer" />
            <name="infant(0-23mth)" datatype="Integer" />
            <name="Type" datatype="String" />
        </Parameter>
    </Request>
    <Response>
        <media-type> txt/html </media-type>
    </Response>
    </Method>
    </Protocol>
</Resource>
</Service>
```

Figure 3.2: Web form described in our schema

Figure 3.2 shows a web form represented in our schema. The resource in Figure 3.2 enclosed within <resource> < /resource> tags has a resource name (*i.e.*, Flight Search), a method name (*i.e.*, POST), a protocol (*i.e.*, HTTP) and a few input parameters along with their data types. A resource graph is a semantic network model which consists of entities and relationships. Entities denote resources which are identified using URIs. Resources may be linked based on the data flow between resources or based on the next transition from the current stage. The data flow relationship is based on the input/output matching between the resources. For example, to retrieve an author resource we need the ISBN of the book the author has written. The transition relation is present in the response and guides a user to the next state. The transition relation is provided by a service provider. The relations recommend new resources, and define the relationship between the resources. We use the relations such as payment as defined in link specification [70]. In addition, we outline some new relations among resources:

1. "seeAlso" recommends another resources. For example a book resource, b1, has see-also relation with book resources, b2 and b3, if all the three books are written by the same author.

2. "sameAs" provides the similar resources. For example, if flight resources, f1 and f2, fly between two cities at the same time, these resources are related by the same-as relation.

3. "isA" defines isA relation between the resources. For example, a single bedroom, and a double bedroom are the resources of room type. Hence both single bedroom and double bedroom have isA relationship with the resource room.

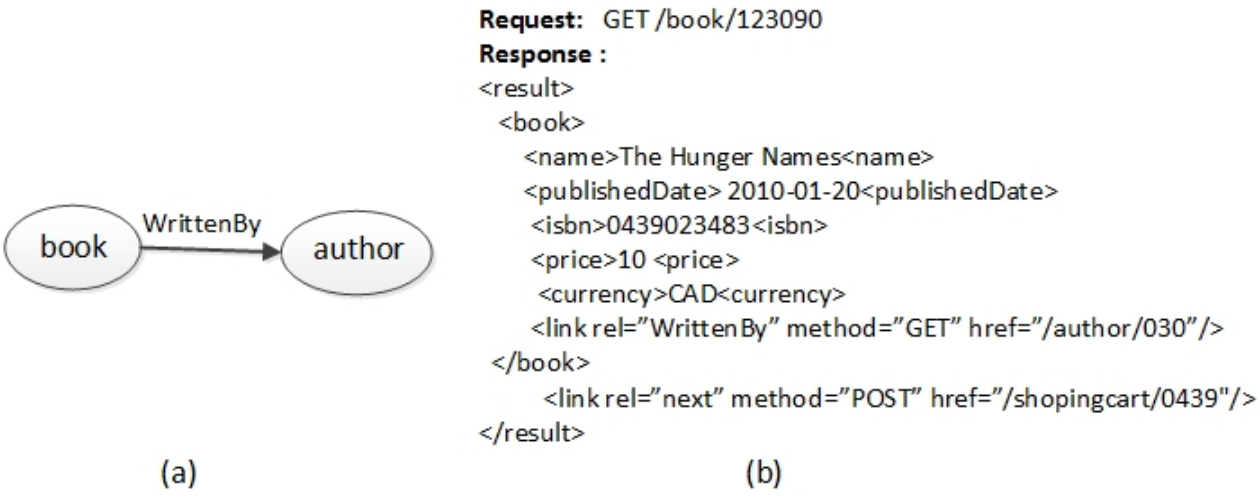


Figure 3.3: Relation between two resources

4. "contains" define different web resources encapsulated in a composite web resource. For example, a search result to a book results multiple instance of the resource books

In addition to these relations, two resources may have a semantic relation which is the relation between resource entities. The resources that do not have the data-flow relation between them, but the concepts between them are semantically related. Figure 3.3 shows the resources and the semantic relations between the different resources. For example, a book resource and an author resource is semantically related. We use a link specification [70] to indicate the relationship between resources. A "rel" attribute defined in the link specification gives information about the semantics of the link. A "method" attribute is added to the link specification to allow the user agent to know the next possible resources to visit along with the method used to visit that resource. Figure 3.3b shows two links with "rel" and "method" attribute.

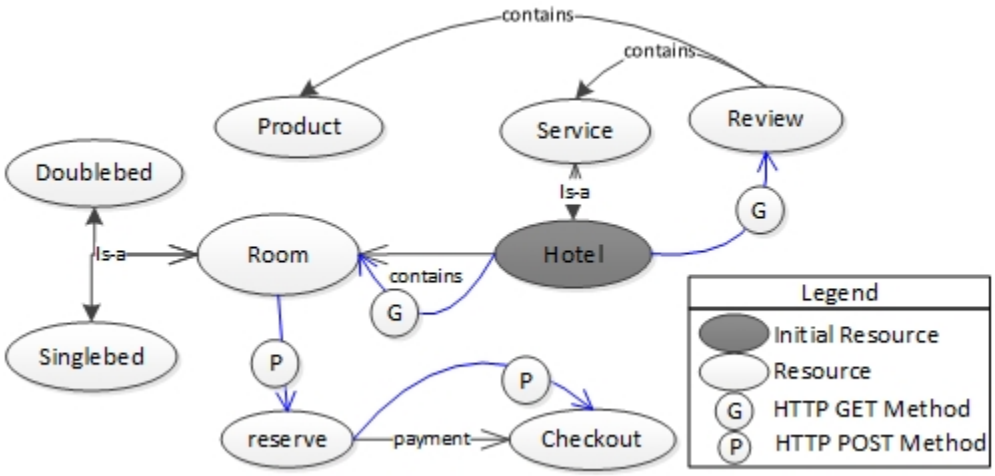


Figure 3.4: Example of resources and their relations

Figure 3.4 shows an example resource graph. The hotel resource is the initial resource. The "review" resource in Figure 3.4 requires the information regarding the resource "hotel". Double bedroom (*i.e.*, "Doublebed") and single bedroom (*i.e.*, "Singlebed") resources have an "is-a" relationship with the "room" resource. All the semantic relations from the "room" resource are carried over to those two different categories of the rooms. The small circle represents the method that can be invoked on resources from the current state. In our resource graph, the initial node denotes a resource from where a user can start consuming a service (*i.e.*, a start point). In Figure 3.4, the "hotel" resource is represented in a darker color as the initial node. When a user requests a service, this node is returned; and from that node, a user can access the resource. This substantially increases the user agents' capability of discovering a resource. For example, if the user wants to book a room, the user needs information regarding the resource room.

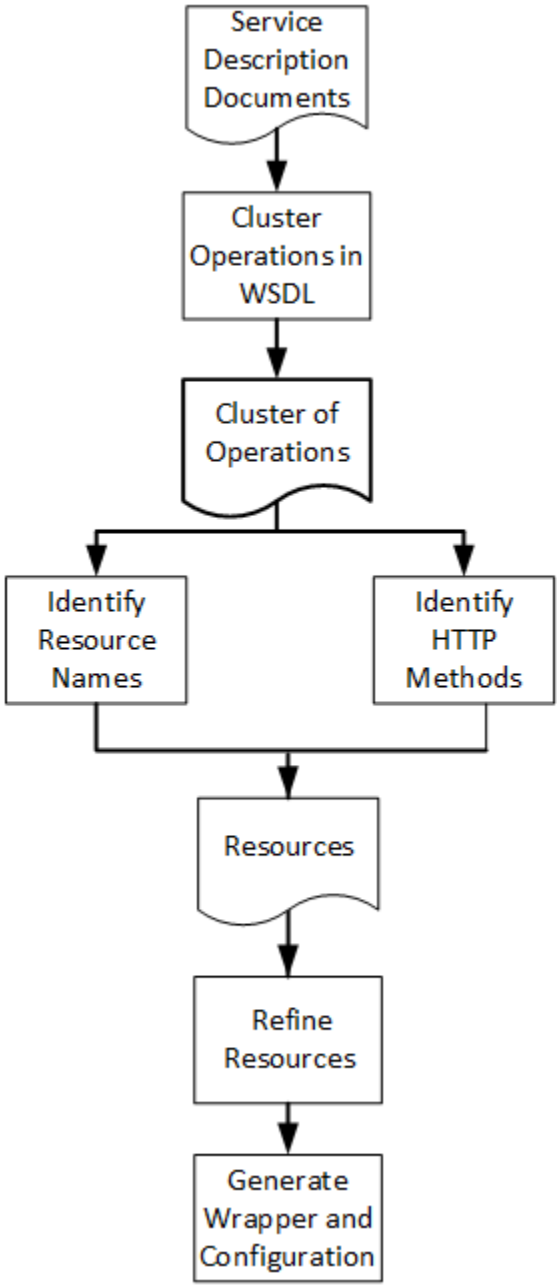
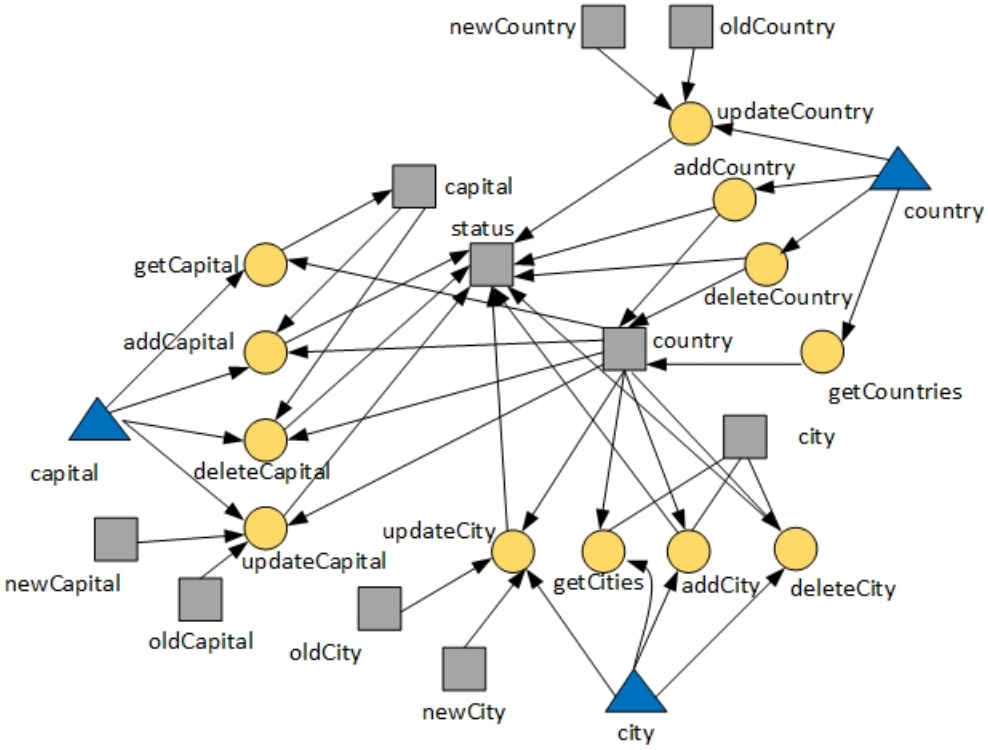


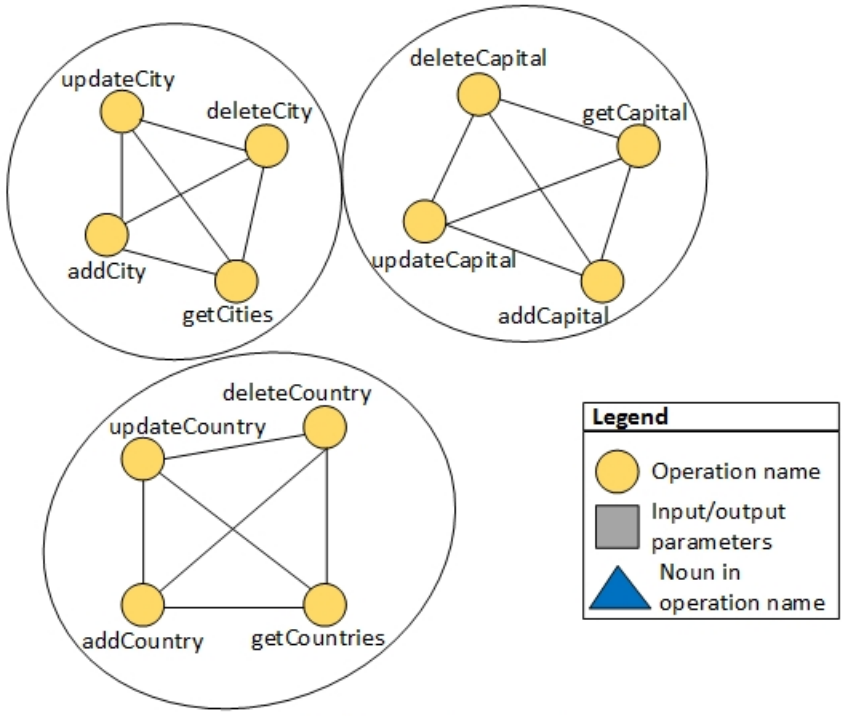
Figure 3.5: Overall approach of identifying resources from SOAP-based services

3.3 Identifying RESTful Service from SOAP-based Services

We identify resources from SOAP-based services by analyzing its WSDL and map the contained operations to resource names and HTTP methods. Figure 3.5 shows the overall steps for identifying resources from SOAP-based services. Initially, we analyze the WSDL document of a SOAP-based service and build a dependency graph which describe the relations between operation names, input and output parameters of an operation defined in the interface of a SOAP-based service. We identify and group similar operations from the dependency graph. Each cluster of operations is analyzed to identify resources and the HTTP-methods associated with the resource. All the modules except the "Refine Resources" module in Figure 3.5 are automated. Furthermore, we manually refine the identified resources and HTTP methods for each WSDL operation. Once the resources are validated and verified by a user, the wrappers and configuration files are automatically generated. The migrated RESTful service is ready for deployment. We describe each step of our approach in identifying resources from SOAP-based services.



(a) Dependency graph



(b) Clusters of operation

Figure 3.6: Clustering operation based dependency graph

3.3.1 Clustering WSDL Operations

Each resource in a RESTful service is associated with four operations (*i.e.*, GET, PUT, POST and DELETE). We need to identify the operations that manipulate the same resource. We build a dependency graph which connects the input and output parameters of each operation along with the semantics (*i.e.*, the nouns found in an operation-name) of an operation. Figure 3.6a shows the dependency graph between the operations, and the input/output parameters. A circle denotes an operation. A square represents an input or output parameter. A triangle means a noun in the name of an operation. For the operations and parameters; the edge pointing to the operation denotes message input and the edge pointing to the parameter denotes the message output.

Cluster analysis is the process of organizing objects into groups whose members are similar in a certain way [54]. We use cluster analysis to group the operations that manipulate the same resource. Each cluster represents a resource. Clustering narrows down the number of resources in a WSDL, as the number of resources is same as the number of clusters. Clustering also helps to identify the HTTP methods related to a resource. We use the semantic distance between the words in operations to cluster the similar operations. The distance metric is defined in Equation 3.1. The similarity between the two operations is the ratio of the sum of the number of common parameters and the common nouns in the name of operations to the sum of the total number of parameters and the total number of nouns in operation names.

$$Similarity(x,y) = \frac{2 * (common_{i/o}(x,y) + common_{noun}(x,y))}{Total_{i/o}(x) + Total_{i/o}(y) + Total_{noun}(x) + Total_{noun}(y)} \quad (3.1)$$

where x, y are the operations. $common_{i/o}(x, y)$ *is the number of common input and output parameters between the operations x and u; and* $common_{noun}(x, y)$ *is the number of common nouns between the names of the operations x and y,* $Total_{noun}(x)$ *is the total number of nouns in the name of operation x,* $Total_{i/o}(x)$ *is the total number of input and output parameters of operation x.*

We group the operations with the value of similarity metric greater than or equal to some threshold value. We choose the threshold values by manually verifying different threshold values for the same WSDL document. Given a dependency graph, we form the clusters of similar operations. Figure 3.6b shows the three different clusters identified from the dependency graph shown in Figure 3.6a. For example, the function "getCapital" in Figure 3.6a takes "country" as an input and returns "capital" as the result. The noun in the name of the function is capital. Similarly, the function addCapital has "country" and "capital" as inputs; and status as the output. Thus, using Equation 3.1 the similarity between addCapital and getCapital is:

$$Similarity(addCapital, getCapital) = \frac{2 * (2 + 1)}{3 + 2 + 1 + 1} = 0.85$$

addCapital and *getCapital* fall above the threshold value, so they belong to the same cluster of operations. We perform a similar approach on all operations and group the similar operations. Based on this criterion the cluster of operations dealing with the same resource is formed as shown in Figure 3.6b. We analyze each cluster to find the name of the resource and the HTTP-method associated with it. The next state is shown by the links between the operations. For each function in a cluster, other functions within the same cluster are embedded as links. For example, when a user calls a service to add a city, the links about updating the city, getting all the

cities of that country and deleting the city are provided to the user.

3.3.2 Identifying Resource Names

Each resource is uniquely identifiable by a URI. Nouns represent the resources. The maximum of four operations (*i.e.*, GET, PUT, POST and DELETE) can be associated with each resource. For each cluster, we identify the name of the resource. Natural Language Processing (NLP) provides the techniques to find the root word by stemming [99]. NLP also provides techniques to identify part-of-speech (POS) to identify lexical categories (*e.g.*, nouns, verbs, and adjectives). We filter the nouns ignoring other lexical forms. The input and output parameters represent the data entities and are considered as nouns. The semantic relationship between the operations, and input/output parameters helps to link words to form the resource. We used WordNet [86] to find a relationship between words. WordNet is a rich set of lexical knowledge. It defines different kinds of relationships between the words.

1. Hypernym represents a "kind of" relation. For example, "car" is a hypernym of "vehicle". A hypernym relation is converted as a subclass relation in the ontology.

2. Hyponym means that a word is a super name of another. For example, "vehicle" is a hyponym of "car". Hyponym is the inverse of hypernym.

3. Holonym describes "whole-part" (*i.e.*, partOf) relation. For example, "city" is a holonym of "country".

4. Meronym is the inverse of holonym and represents a part-whole relation. For example, "window" is a meronym of "building".

Table 3.1: Rules to decompose words

Rule	Name	Words
Case changes	FindCity	Find, City
	addCountry	add, Country
Suffix containing number	City1	City
Underscore separator	Find_city	Find, City

Table 3.2: Semantic, input parameters and output parameters in a cluster

	getCity	addCity	deleteCity	UpdateCity
Semantic	city	city	city	city
Output Parameters	city	Status	Status	Status
Input Parameter	country	City, Country	City, Country	Country, oldCity, NewCity

For each cluster as shown in Figure 3.6b, we separate the words in the operation name, remove the stop words (*e.g.*, a, an, and the), use stemming to find the root word and identify the lexical categories to filter the nouns. Each cluster is categorized into three categories: 1) nouns used in the name of the operation (we name it semantic-relation), 2) the input parameters, and 3) output parameters of the operation. We apply the rules in Table 3.1 to decompose the words in the operation name.

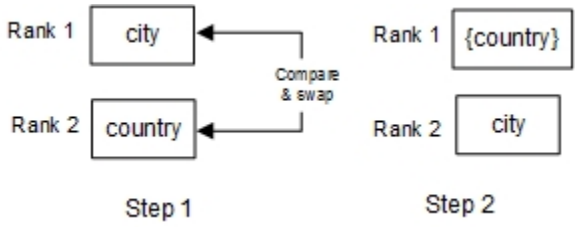


Figure 3.7: Ranking words to identify the resource name

From the initial set of words, we choose the common words from the semantic part of the operation, and the input/output parameters. For example, in Table 3.2, we consider the cluster {addCity, updateCity, getCities, deleteCity} shown in Figure

3.6b. The operations include a common word, *i.e.*, "city". The input parameters have the word, "country" in common. Therefore, the initial set of words is {city, country}. We arrange these words to form a resource name. The organization is done on the basis of the word's position in the name of the operation, and the semantic relations between the words. For example, parental words (*e.g.*, country in {country, city}), words that already are resources are given higher priority as shown in Figure 3.7. The following steps rank the elements in operation-name and input-output parameters.

1. Rank the words according to the position of occurrences. Words in the semantic region are given the highest priority, input parameters as second and output parameters as third.

2. Check if the word is already used to name a resource. If so, such a word is given a higher priority.

3. Check the semantic relations between the words and prioritize the words accordingly.

4. Combine the words in the initial set to form the resource.

For the name of the operation that are not able to categorize, we put the name of operation in the semantic region and form the URI for the operation as /operation-name/{input} and turn the flag for editing. The editing flag helps the user to distinguish such resources before generating RESTful services.

In case of more than one word for the same position in the resource name hierarchy, we examine the semantic relations between the words and their frequencies in the WSDL document. For example, if two words exist as a resource with the same

priority, we use semantic relations between the words to determine their positions, and if no relation exists, then we order the words according to their frequencies of the word in the WSDL document. To illustrate the process, let us take a cluster in Figure 3.6b. Table 3.2 shows the semantics and the input/output parameters in the cluster. The common elements among them are {city, country}. A city is a-part of a country. Therefore, there is a semantic relation between "city" and "country". Hence, "country" comes before "city" in the resource name. Since "country" occurs in input parameters, we denote "country" within {} making it one of the input parameters. Hence, the resource for this cluster is {country}/city.

3.3.3 Identifying Resource Methods

In this step, our approach automatically identifies the legitimate HTTP-methods for each resource. Ideally, each cluster should contain four operations that are mapped to a single resource and four different HTTP-methods. Resource methods are identified by analyzing the verb-part of the operation name. We identify the HTTP method associated with the operation by analyzing the semantics of the operation name. If HTTP methods cannot be identified by analyzing the semantics of operation names, we calculate the fan-in and fan-out of the operations. Fan-in and fan-out indicate the number of the parameters taken as inputs and the number of outputs given by the operation. If the ratio of fan-in to fan-out is less than 1.0, we assume the method denotes the modification or the creation of a resource and further encourage the user to validate the judgment.

In the case where two operations in the same cluster have the same HTTP method,

we revisit the resource identification to check if the cluster contains two different resources. For example, in case of two operations "getLatestBlogs()" and "getBlog()", our approach may recognize the same resource blog and the same method GET. In such cases, we revisit the resource identification process and include the excluded part in the semantics of the operation name called "latest" to form the new resource "blog/latest" for "getLatestBlogs()". If a different resource cannot be formed, we tunnel the operation through the POST method. POST is considered unsafe and idempotent; hence it is the best to use this method when we are not sure which method can be associated with the resource. Such resources are flagged so that it can be clearly identified in the user interface and allows the service provider to modify it. For example, in Figure 3.7b, the cluster {getCity, addCity, deleteCity, updateCity} contains the resource identified as $\{country\}/city$. We analyze the semantic meaning of the operation and determine the corresponding resource methods (*i.e.*, GET, POST, DELETE, and PUT). There is no conflict. In that case the perfect match is found, and hence fan-in and fan-out of operations are not calculated.

In some cases, when our approach cannot identify the resource methods automatically we use POST as the preferred method and include the SOAP operation name as one of the parameters in the URL. We flag such results to increase the visibility of such resource while validating the results. We highlight such flagged resources with a highlighted color in our interface. Our approach is more suitable for data-oriented web services. Data oriented services consume and retrieve data based on a user's request. Any computation can be a resource such as, word processing, image manipulation and temporal service (*e.g.*, weather in Kingston over the next four days). The computational services have many operations associated with the same object.

For example, in an image manipulation service, a "POST" operation was mapped to multiple methods such as re-size picture (*resizePicture*),and sharpen color (*sharpenColor*). We found that the operations in computation-oriented web services are difficult to map to RESTful services. The generated resources from the computation-oriented web services need more manual intervention as the most of the operation names contain multiple similar verbs and hence most of the resources are tunneled through the POST method.

3.3.4 Generating Wrapper for Service Migration

Our approach enhances SOAP-based services, giving them the capability of RESTful services. Our approach does not affect the operation of traditional services. Instead, all the RESTful requests are converted to corresponding SOAP-based service operations. We convert messages back and forth. We parse the SOAP message. We exclude the SOAP-envelope and include the data in the REST output representation. A RESTful client's inputs are converted to the SOAP-based message format. Correspondingly, the output from a SOAP-based service is converted into one described in the content-type attribute of the HTTP request header. The Link formation is also done at this stage, which guides the client to the next state. Figure 3.8 shows the framework to convert the message passing between a SOAP-based service and RESTful clients.

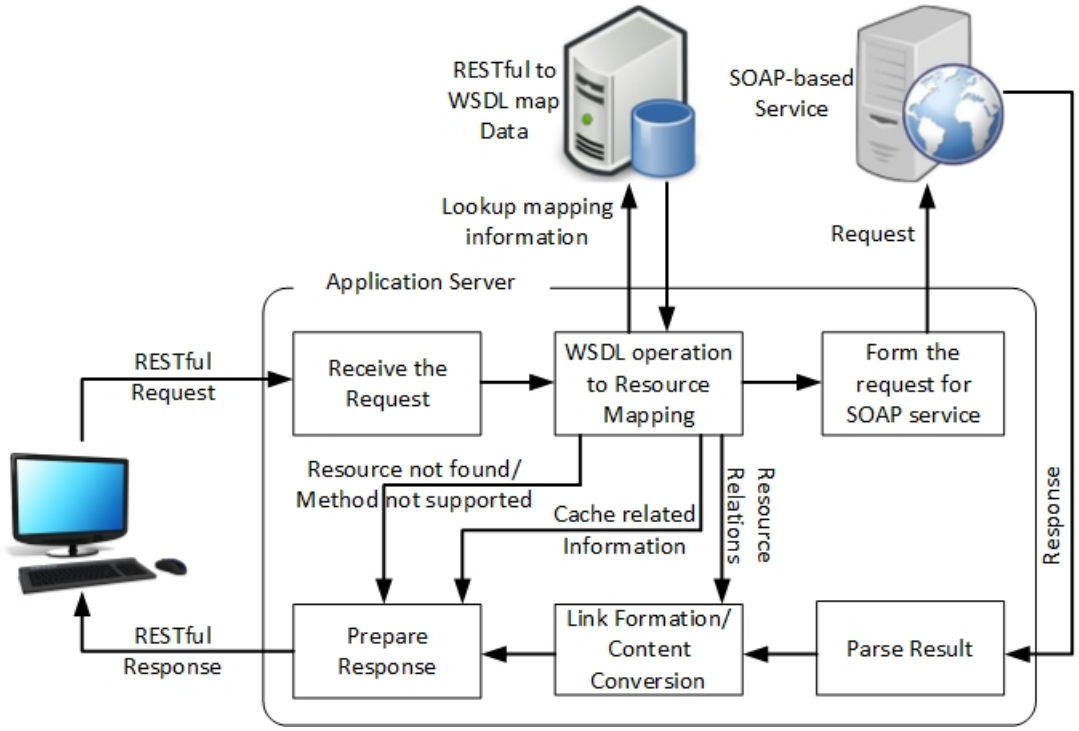


Figure 3.8: Process of transforming messages between SOAP-based services and RESTful clients

Our proxy captures the REST request for a resource. Based on the resource URL and the HTTP method, the proxy finds the corresponding SOAP operation using the "WSDL operation to Resource mapping" module. The mapped SOAP-based operation is invoked and the received result is parsed and changed to a response format as requested by the client in the content-type header. The framework uses the standard HTTP-Error messages if an error occurs. For example, if a user uses an HTTP-method that is not supported by a resource our approach generates an error called "405 Method Not Allowed". The "Prepare Response" module of our framework is responsible for a proper response code and cache related headers in the response. This module also parses the SOAP-response, removes the unnecessary SOAP-envelope. The modules also generate the links by adding resource relationships.

The link relations help a user navigate from one resource to another. All this process is done automatically.

S.No	Action		Resource	HTTP Method	WSDL Method	Details	Links To
1	Edit	Remove	/methods	GET	GetMethods	Details	2 5 8
2	Edit	Remove	/domain/members	GET	GetDomainMembers	Details	3 1 5 8
3	Edit	Remove	/domain/names	GET	GetDomainNames	Details	2 1 5 8
4	Edit	Remove	/customers/{id}	GET	GetCustomerByID	Details	5 6 1 2 8
5	Edit	Remove	/customers	GET	GetCustomers	Details	6 8 1 2
6	Edit	Remove	/customers/modified	GET	GetModifiedCustomers	Details	5 1 2 8
7	Edit	Remove	/service/location/{servloc}	GET	GetServiceLocationByServLoc	Details	1 2 5 8 9
8	Edit	Remove	/service/location	GET	GetServiceLocation	Details	1 2 5 9
9	Edit	Remove	/service/location/modified	GET	GetModifiedServiceLocation	Details	1 2 5 8

Add New Resource

Confirm

Generate WSDL 2.0

Generate WADL

Figure 3.9: Screen-shot for listing predicted resources

3.3.5 Prototype for Identifying RESTful Services from SOAP-based Services

Ambiguities among the resources are removed in this stage. Such ambiguities include more than one resource competing for the same name or HTTP methods. In addition, our approach may misidentify the resource or HTTP-methods. A user can manually validate and modify the resource, HTTP method and cache related information. HTTP has the standard way for cache validation using Entity tags (*i.e.*, E-Tag) which allows a client to make a conditional request. The service provider

may additionally set a header called "expires" to denote the content is stale after the given time frame. We provide a simple Graphic User Interface (GUI) where the user can examine the resource names and methods along with the corresponding WSDL operations. Figure 3.9 shows a screen-shot of the prototype. The GUI provides the functionality to generate the WSDL2.0/WADL description of the corresponding RESTful service. The resources that do not exactly match and are tunnelled through the HTTP POST operation may need to be modified. We highlight all the resources that are tunnelled and have inappropriate naming. Moreover, the GUI also shows the links to other resources that a representation contains.

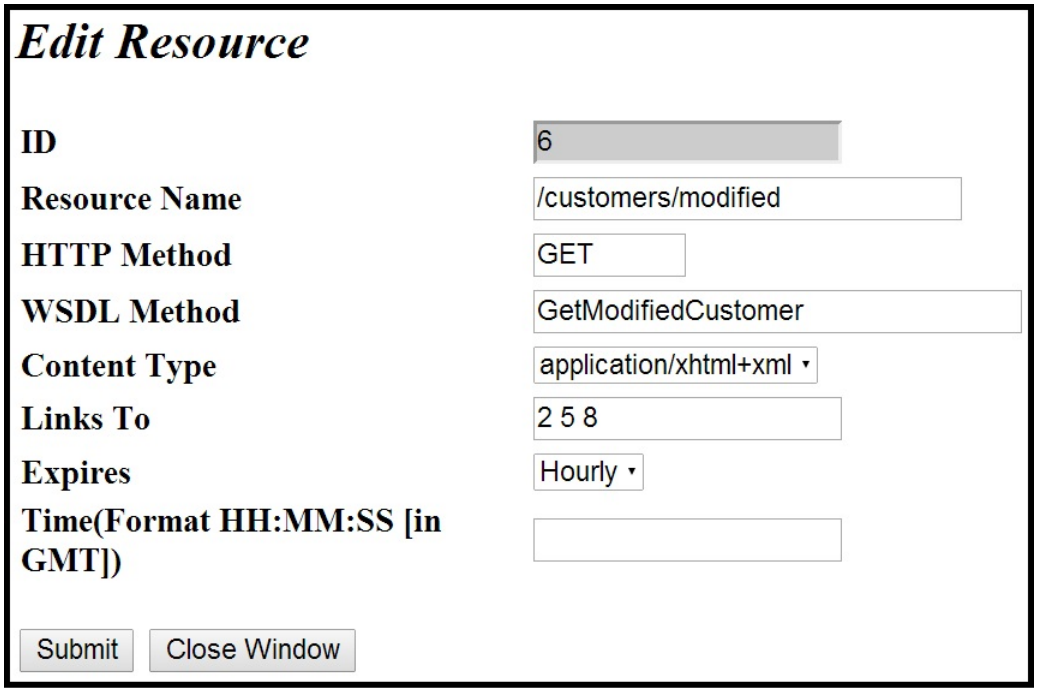


Figure 3.10: Screen-shot for editing a predicted resource

A user can choose to change the different attributes associated with a resource such as resource name, HTTP method and content type except the resource id as shown in Figure 3.10. Once a user confirms, we generate the corresponding service

description files. If the service provider wants the resource to be different, the GUI can be used to edit the resource name and HTTP methods. The name within {} has to match to one of the input parameters. Figure 3.10 shows the user interface where a user can edit all the fields except the ID of the resource. If the service provider has a cache-related requirement, this requirement is fulfilled at this stage. We present an approach to describe caching terms in terms of hour, day, week, month and year along with the time of a day. The service provider can also modify the link to other resources. Once all the information is verified by confirming the resource names, and the HTTP methods, we automatically generate necessary configuration files, servlets, and wrappers. The RESTful service is ready for deployment.

3.4 Modeling User's Tasks in Web Applications

A task is a collection of resource interactions to accomplish a goal. Task identification is centered on the question: What will a user do with a web application? A task is a course of actions that a user performs on a web application. A task captures a navigation structure of a web application that performs a certain functionality.

A task is identified on the basis of two main characteristics: 1) should be reusable, and 2) should perform a functionality. An example of a task includes searching a product, login into a web page and purchasing a product. For example, by identifying a task to buy a product, service providers can reduce multiple user interactions (e.g., product selection, credit card verification and address confirmation) to one task where a user does not have to go through multiple interactions. The types of resources included in tasks describe the nature of a task. We have identified three types of resources used in web applications as shown in Table 3.3. The classification is based

Table 3.3: Different types of resources in a web application

Resource Type	Description	Example
Type I	Type I resources represent a simple process without any input parameters	Information web pages, such as a web page without any parameters.
Type II	Type II resources take input parameters and output representation	Searching products based on keywords, such as Amazon product search web page.
Type III	Type III task is a complex process with input parameters, output representation and client side scripts.	A login process with client-side validation, such as an email login process.

on how a user-agent (*e.g.*, a web browser) interacts with a resource. Type I resources have fixed URLs. However, the content may change over time or when a user invokes a URL. An example of a Type I resource is a weather page [129]. Type II resources take URL parameters or payload as input. The representation of a resource is updated with the changes in the parameters. An example of a Type II resource is a product search page of an e-commerce site [35]. Type III resources use both input and client side code to manipulate the changes in resource states. A user event, such as a button click calls a JavaScript function. The HTTP protocol is invoked from the JS function. An example of a Type III is a login page that validates the format of a user input before requesting a resource [50].

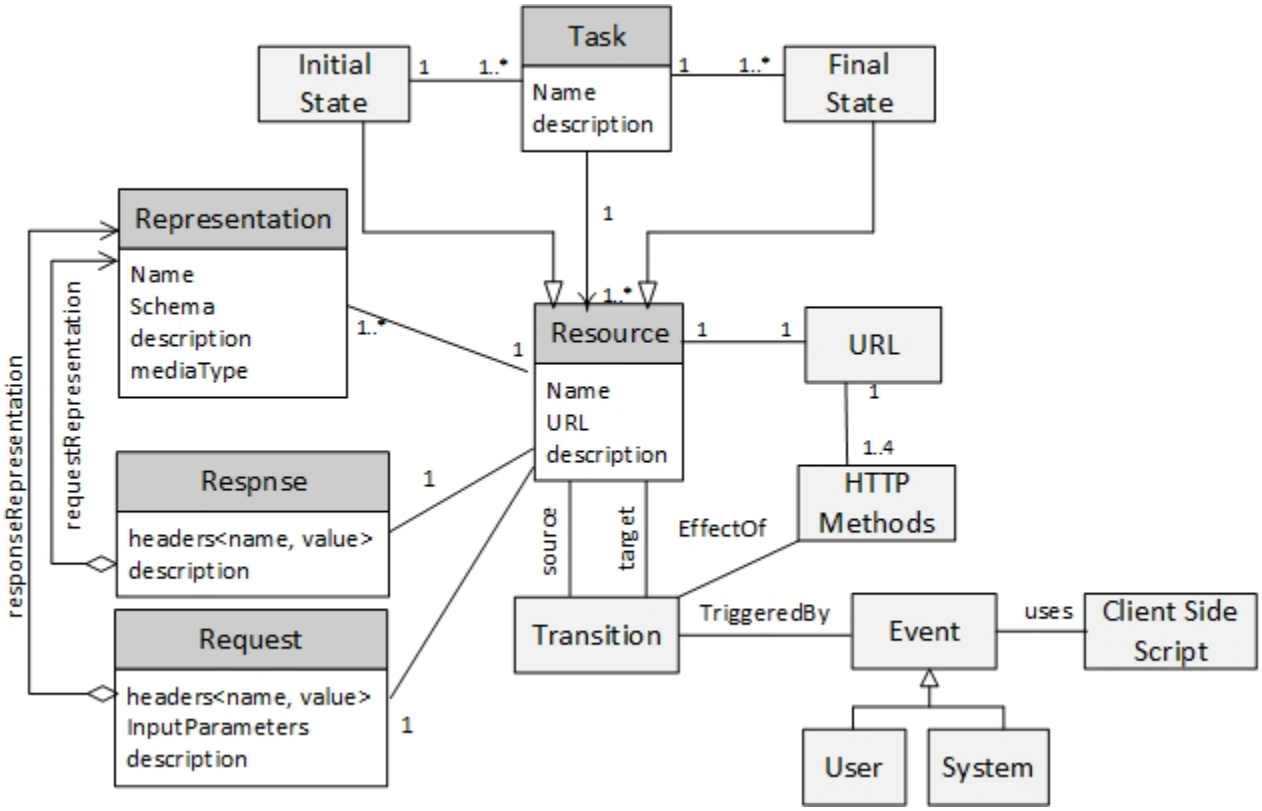


Figure 3.11: Meta-model for users' tasks

Figure 3.11 shows the meta-model to represent users' tasks. A task starts with an initial resource (*i.e.*, initial state) and ends with one or more final resources. Each resource has a URL, HTTP methods, requests and responses. The requests and responses include header information, input parameters and response representation. A response representation is the description of the messages sent to or received from a resource in terms of a technological language. Currently XML and JSON are the most popular languages for describing these messages. Therefore, a representation is defined in the meta-model as an abstract entity that is generalized in the different types of representations according to the corresponding media-types. For accessing or modifying a resource, one of the four HTTP methods (GET, POST, PUT or

DELETE) is used. HTTP headers define the operating parameters of a resource interaction. While completing a task, a resource undergoes a series of transitions. A transition can be triggered by a user action (*e.g.*, form submission and resource request) or by system events (*e.g.*, automatic updates of the representation at certain intervals and web page redirections). Resources in our meta-model can be one of the three resources listed in Table 3.3. Based on the model presented in Figure 3.11, we describe each task as a RESTful service.

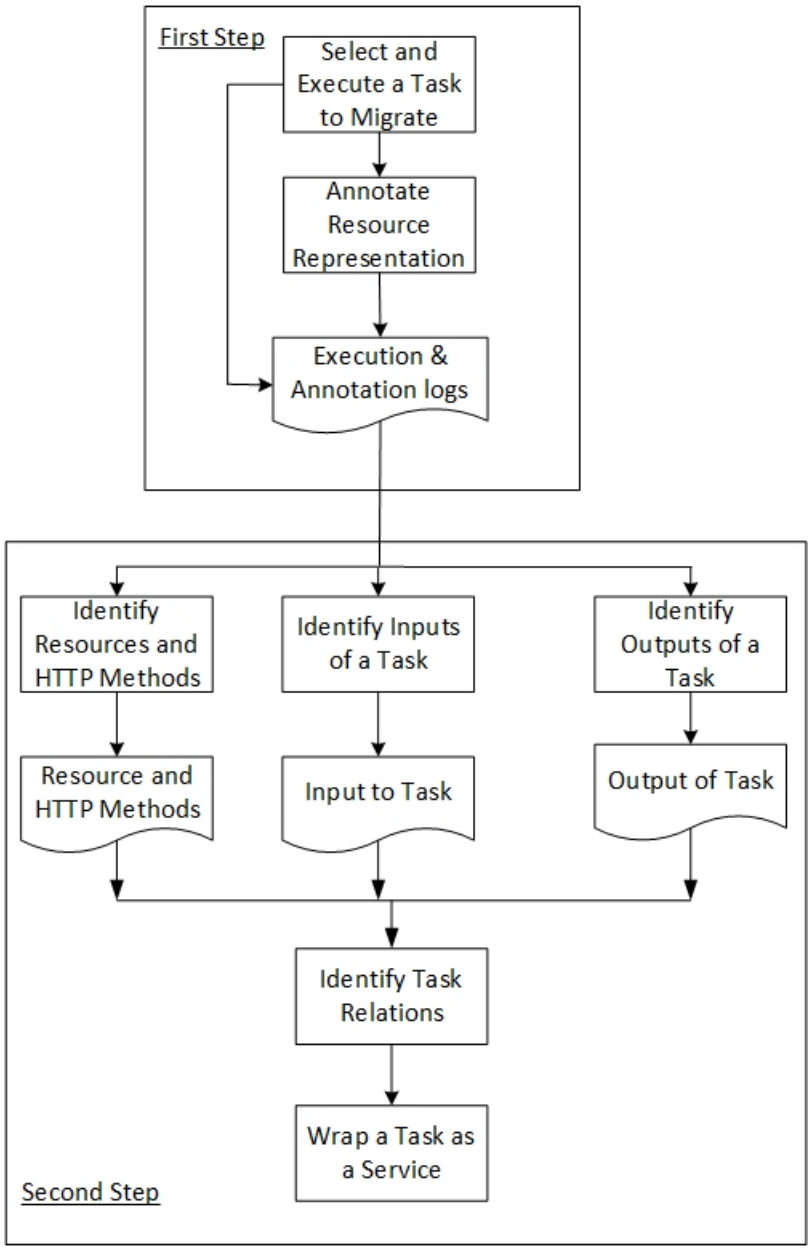
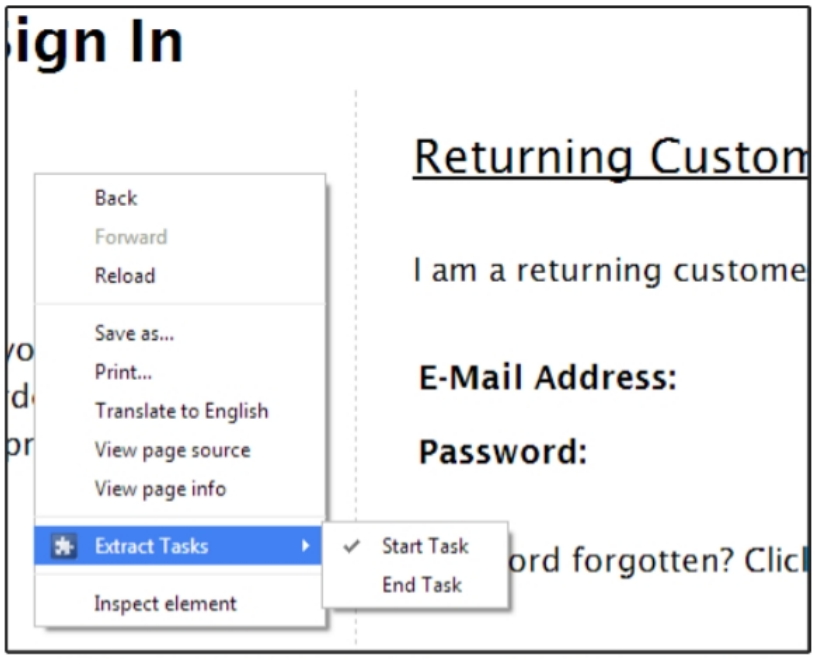


Figure 3.12: Overview of our approach to identify resources from a web application

3.5 Identifying RESTful Service from Web Applications

In this section, we discuss our approach to identify resources from a web application. In this work, we consider a web application as a number of tasks. We represent each task as a resource. We reverse engineer the client representation, request and response pattern of a web application, to extract tasks as RESTful services. A task is a goal specific functionality such as "search a restaurant", and "reserve a table in a restaurant". A goal may be defined as a state of affairs that a user wishes to achieve; a task is the course of action a user goes through in order to achieve this state. In a web application, the code responsible for a task is scattered between several files, and it is written in different languages, such as service side scripting languages (*e.g.*, PHP), database query languages (*e.g.*, SQL), client side scripting languages (*e.g.*, JavaScript), and HTML. We treat web applications like black-boxes and extract tasks by analyzing client-side representation of a web application. Extracting a task from a client representation can be particularly useful when the code-base of a web application is not available. The extracted tasks can then be specified in terms of a RESTful service and deployed through proxies accessing the original web server and parsing its responses.Figure 3.12 shows the overall process of our approach to identify RESTful services from web applications. Our approach is semi-automated and requires a user's input at various stages of the service extraction process. The user does not necessarily need to be an expert in the language and technology used to develop the web application. However, s/he needs to have enough knowledge about the web application and should be able to identify a reusable task. Our approach consists of two steps as shown in Figure 3.12. The first step of the process is to select and execute a task to migrate. This first step is manually done. The selection of a

task is based on different factors such as reusability, its business value and its state independence. We leave this as a design decision on a user. A user can select any task to migrate as a RESTful service in a web application. We provide a tool to indicate the start and the completion of a task. A user runs a web application multiple times denoting the start and the completion of tasks to capture all possible scenarios.



(a) Menu to denote start and completion of a task

(b) Annotated content in a web page

Figure 3.13: Different phases of task identification

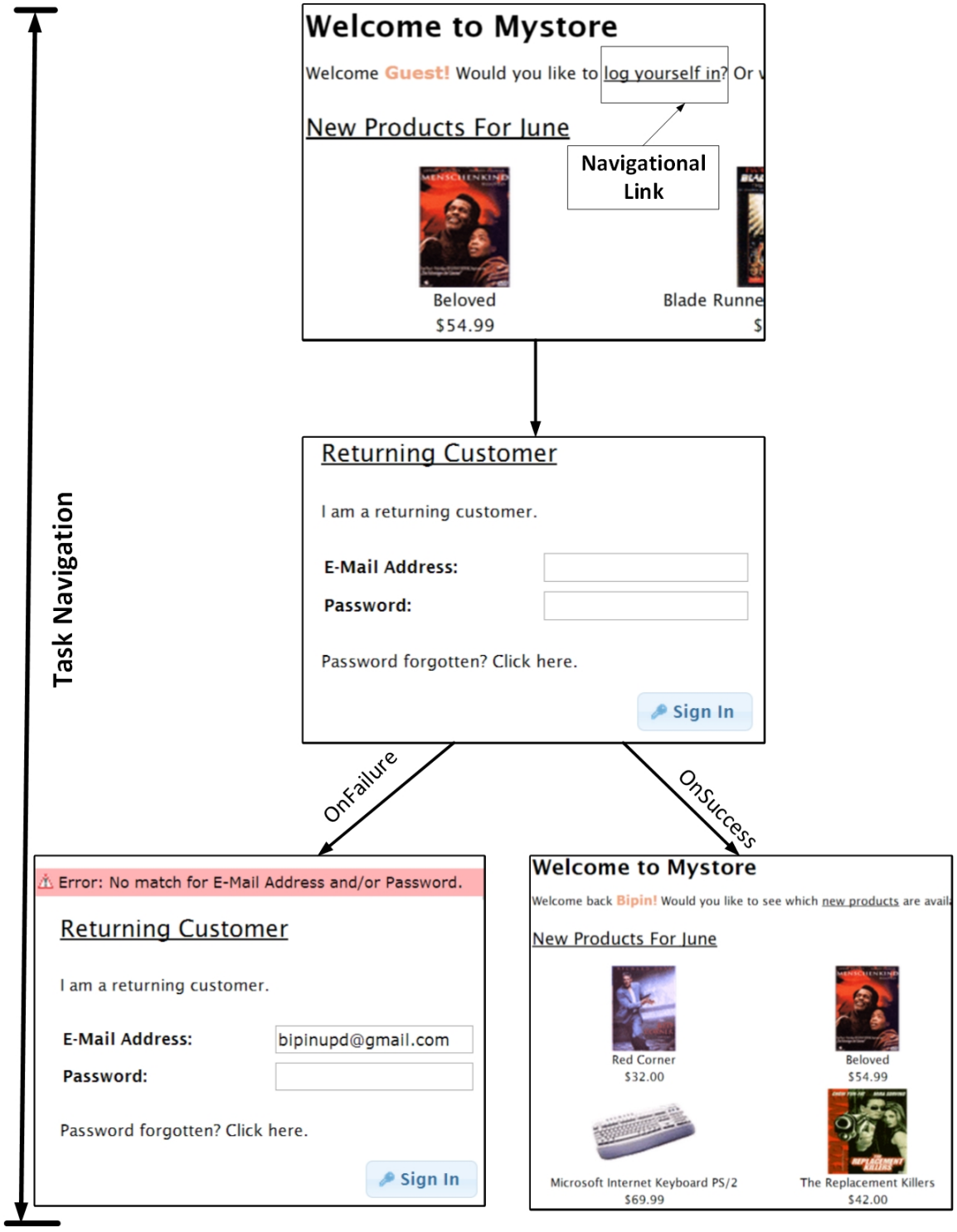


Figure 3.14: Task involving multiple web page navigation

Figure 3.13a shows a menu to denote the start and the end of a task. In addition, we provide a tool that allows a user to annotate the fragment of representation as shown in Figure 3.13. This annotated fragment is analyzed to identify the output of a task. The second step is the analysis of the annotation logs and execution logs to determine input, output and HTTP methods and resource relation. We capture all HTTP interaction during a task completion process e.g., Figure 3.14 shows web pages involved in a login task. In the rest of this section, we discuss the different phases of RESTful service extraction as shown in the second step of Figure 3.12 along with how these data are captured.

3.5.1 Identifying RESTful Service Inputs

A resource transition can be in one of the two forms *viz.*, user event or system event. Web forms and hyperlinks are the most typical ways to provide input to a web application. A web form submission does not always invoke a resource. A web form generates a number of events. These events are handled by client side JavaScript functions. JavaScript programs are executed by the client's web browser and have access, via a document object model, to the resources of the browser, in particular, to the HTML document shown in the browser. For this purpose, the document is represented as a hierarchical object structure where the attributes of each object can be accessed or manipulated by the standard "dot notation". For instance, the class identifier (whose meaning is usually defined in a style sheet) of an object element in an HTML document can be changed to *myStyle* by the assignment $elem.className =$ "*myStyle*". JavaScript programs are usually executed by the web browser when some event occurs. For instance, if an input button in an HTML form

has an attribute *onclick* = "$fun(x)$", the function call $fun(x)$ is evaluated whenever the user clicks this button. Our tool keeps track of all the events generated during the completion of a task.

The web form and hyper-links contain semantic information (*e.g.*, labels). It is challenging to identify labels that describe HTML input elements. Especially, web forms may have different layouts. The positions of labels in a web form depend on the designer of the web page. Labels can be placed above, below, to the left, or to the right of an input element. To determine the label representing an input element, we analyze the contents of a web page delimited by the opening and the closing tags of an HTML partitioning element that separates the different sections of a web page. For example, paragraph tag (*i.e.*, <p>) separates a paragraph in HTML. The text nodes under the partitioning element are part of the same blob (*i.e.*, a text contained within a partitioning element). However, style tags, such as the italic tag (*i.e.*, <i>) and the bold tag (*i.e.*, <b>), add styles (*e.g.*, bold and italic) for the text. Therefore, style tags are not considered as partitioning elements.

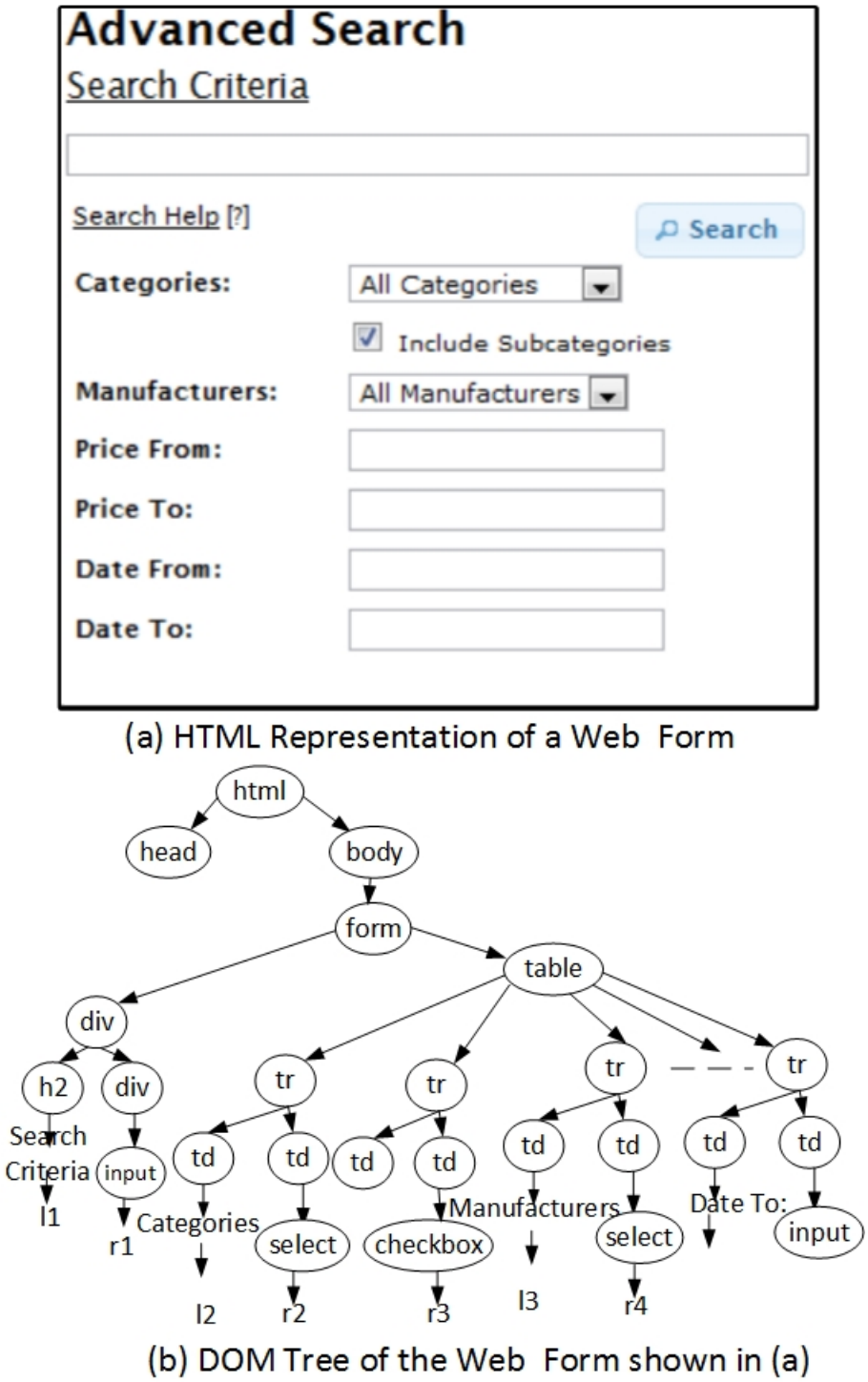
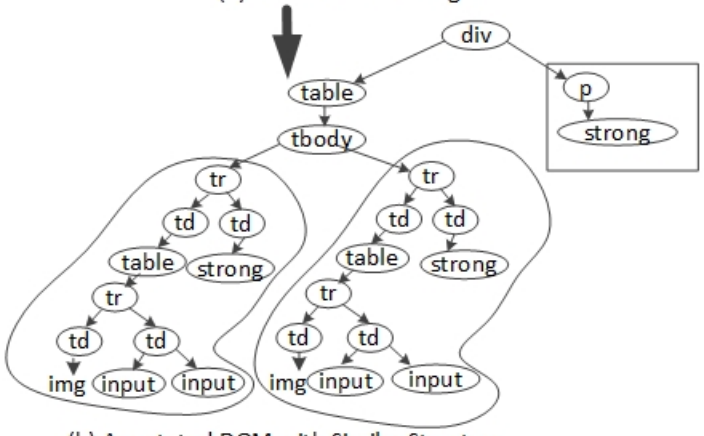


Figure 3.15: HTML and DOM representation of web query interface

A web form is designed as a hierarchy of HTML tags. Labels and form input elements are often positioned in proximity in the hierarchical structure of tags. Hierarchically nested labels and form input elements are placed close to the lowest common ancestor in the hierarchy. If a label and the associated input element are in the same parent structure, they are close to each other within the parent structure. The hierarchical proximity between the elements helps to associate the input elements with the text blob. Figure 3.15a shows a screen-shot of a web query interface. Figure 3.15b shows a fragment of the DOM tree of the query web form shown in Figure 3.15a. In Figure 3.15b, the input field *r1* is in closer hierarchical proximity with the label *l1* (*i.e.*, "Search Criteria") than the label *l2* (*i.e.*, "Categories"). Therefore, the label *l1* (*i.e.*, "Search Criteria") should be associated with the input *r1*. To identify the association between the input element and labels, we traverse and analyze the DOM tree to find the text nodes that constitute a label. When a partitioning element (*e.g.*, paragraph tag <p>) is reached, a new label is created. The text node under the partitioning element is added to the label. If the partitioning element contains another partitioning element as a child, then the text nodes that appear under the sub-partitioning child belong to the text blob of the sub-partitioning child. For each input element, we compare the hierarchical proximity between the input element and the text blob. The distance between the text blob and the input element is given by the number of nodes to visit the text blob from the input element. The label with the least distance from the input element is considered as a candidate of an input description. For the example shown in Figure 3.15b, the distance between the nodes *r1* and *l2* is 2; the distance between the nodes, *r1* and *l3*, is 4; and the distance between the nodes, *r1* and *l4*, is 5. The node *r1* has the least distance with the text blob *l2*, and

hence the node *l2* is selected as the description tag for the node *r1*. If more than one candidates are identified, we calculate the edit distance [83] between the candidates and the "name" attribute of the input element to choose the candidate for the input description tag. The edit distance between two strings of characters is the number of operations required to transform one string of characters into the other. The input is not just related to the HTML links and forms. Web browser cookie technology provides persistent data storage on the client side. A cookie is a data set consisting at least of the cookie's name, value and domain. It is sent by the web server as part of an HTTP response message using the Set-Cookie header field. The cookie's domain value is used to determine in which HTTP requests the cookie is included. Whenever the web browser accesses a web page that lies in the domain of the cookie, the cookie is automatically included in the HTTP request using the cookie field. Cookies are often used as authentication tokens by today's web applications. After a successful login procedure, the server sends a cookie to the client. Every following HTTP request that contains this cookie is automatically regarded as authenticated. We track the changes in the cookies and other HTTP header fields.

(a) Annotated Web Page



(b) Annotated DOM with Similar Structure

XPath	Name	Example
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]//table//tr[1] /d[1]/table/tr[1]/td[1]	Name	Speed, Samsung Galaxy Tab
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]/table//tr[1] /d[1]/table/tr[1]/td[1]/img	Picture	
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]/table//tr[1] /d[1]/table/tr[1]/td[2]/input[1].value	product_id	17 ,28
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]/table//tr[1] /d[1]/table/tr[1]/td[2]/input[2].value	cart_quantity	1,1
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]//table//tr[1]/td[2]	price	$39.99, $749.99
//*[contains(concat(" ", @class, " "), concat(" ", "contentContainer", " "))]//p/strong	sub-total	$789.98

Submit

(c) Extracted XPath with name of the elements

Figure 3.16: Identifying the data segment in the HTML representation

3.5.2 Identifying of RESTful Service Outputs

HTML pages returned by a web application are generated by a common template and the syntax and the semantics in a web application are fixed. Though a typical web page contains different segments and information, only a particular segment of the web page contains results related to a user task. When a user submits a form, two kinds of data are received a) header information with status code, and b) resource representation. Our approach keeps track of all the changes in the header fields. We provide a browser based annotation tool that helps a user to select a region of an HTML representation of the output of the task. We perform the following steps on the selected region of representation.

1. Select a portion of the HTML representation using the tool. Figure 3.16a shows an example of an annotated HTML page.

2. Find the starting and ending positions of the selected region.

3. Parse the HTML of the source document and select the starting and ending positions identified in step 2.

4. Identify regions with similar DOM structure. Similar DOM structures represent similar data segments. Figure 3.16b shows two similar DOM structures. We use the following three heuristics to identify the semantics of the identified elements.

   (a) Match web form labels in the response. We locate if any labels discovered from a web form are present in the response page.

   (b) Search for labels in table headers. The HTML specification defines tags such as header cell in HTML table, <TH>, and header content in HTML

table, <THEAD>, list the columns of HTML tables; and

(c) Search for voluntary labels encoded in the response page. For example, if the response page may contain a column with '$', then the data item represents currency related field such as price.

Our approach identifies and refines the semantics of the extracted data template. Figure 3.16c shows a screen-shot of GUI that helps a user to refine the extracted data template.

3.5.3 Identifying Resource Names and HTTP Methods

In this step, we identify the resources used to accomplish a task and the execution sequence between those resources. During the course of completion of a task, different resources may be invoked using different HTTP methods. We always select unsafe method over safe method and idempotent method over the un-idempotent method. For example, if a task uses two resources using GET for one resource and POST for another, the HTTP method for that task is POST.

Similarly, based on request/response headers, we assign the headers for the task URL. For example, the "If-Modified-Since" header takes the lowest date among all the dates used in a resource. If the "Set-cookie" and "Cookie" headers change multiple times during a task completion phase, the most recent change in the cookie is propagated to the client.

3.5.4 Identify of Resource Relations

We model a task as a RESTful service; hence we include Hypermedia As The Engine Of Application State (HATEOAS) [42] behavior in extracted services. The

HATEOAS behavior guides a user through different tasks. We use two different approaches to obtain HATEOAS information between extracted tasks. The first approach is based on how a user navigates through web pages during a task. For example, after finishing a "register" task a user performs a "login" task. Hence, the representation of "register" resource should contain "login" resource. In addition to that, we analyze all the extracted links and forms in HTML representation that help to identify the possible next state information. Based on the analysis of representations, we define rules to extract the resource. We analyze all the web forms, incoming and outgoing links between the tasks to identify task relations. We propose the following rules to extract task relations from a client-side representation.

Rule 1: Identify state changes without request and response

This rule identifies the client side script that does not perform HTTP requests, and responses, but changes the state for an HTML representation. In such a case, the URL, HTTP-methods, and parameters between the two resource interactions remain the same, whereas there is a change in the DOM elements. This change is due to the client side scripts, such as validation of data entered in a web form. For example, in Figure 3.14, when a user tries to submit a form without user name, and password, the representation displays an error, but the validation is only performed in the client-side. Hence, the state of the resource representation is dependent on the client-side script.

$Resource_a = \{URL_a, HTTP - Method, Parameter_a, Representation_a\}$

changes to

$Resource_a = \{URL_a, HTTP - Method, Parameter_a, Representation_b\}$

$where, Representation_a \neq Representation_b$

The change in the HTML representation is triggered by client side scripts. For example, client validation of web forms such as user registration and data entry.

Rule 2: Identify related resources

A web developer embeds the links in web pages to help a user to choose the next step. We cluster URLs with similar parameters and resource paths. For example, if the URL of a "product info" resource is http://foo.org/product?pid=xx and the URL of a "product review" resource is http://foo.org/review?pid=xx, the parameter names in the URLs of the "product info" resource and the "product review" resource are similar and belong to the same cluster. We also identify the next resource a user can perform after completing a resource. We extract all next-state elements (*i.e.*, "ExtractFormsLinks"). For two given resources, we choose non-recurring elements (*i.e.*, $Next_{TA}$ and $Next_{TB}$ as shown illustration). A non-recurring element is unique elements between a set of next-states. We identify resources are present in the non-recurring elements list. For example, in Figure 3.14, after a user "logs in" the "log off" link appears in response representation. Hence, "logs in" task response representation should contain the link to perform log off task.

$Next_a = ExtractFormsLinks(Resource_a)$

$Next_b = ExtractFormsLinks(Resource_b)$

$Next_{common} = Next_a \cap Next_b$

$Next_{TA} = Next_b - Next_{common}$

$Next_{TB} = Next_a \cup Next_{common}$

Rule 3: Identify dependent resources

For two resources (*i.e.*, resource A and resource B), if the completion of resource A is required before starting resource B, resource B is dependent on resource A. When an execution flow of a task is present in another task, dependent relation is discovered. For example, "checkout" of a shopping cart resource needs the "logs in" resource to be invoked first, "checkout" resource is dependent in "logs in" task. This rule identifies the client side script that does not perform HTTP request and response, but changes the state of HTML representation.

$Resource_a = \{URL_a, HTTP - Method, Parameter_a, Representation_a\}$

$Resource_b = \{URL_b, HTTP - Method, Parameter_b, Representation_b\}$

$Resource_a$ redirects to $Resource_b$

At the end of this stage, every resource is denoted by a resource URL, HTTP methods, input parameters, output parameters and resource relation links. Figure 3.17 shows a visual representation of product search example. For simplicity, we hide the resource URLs and HTTP methods.

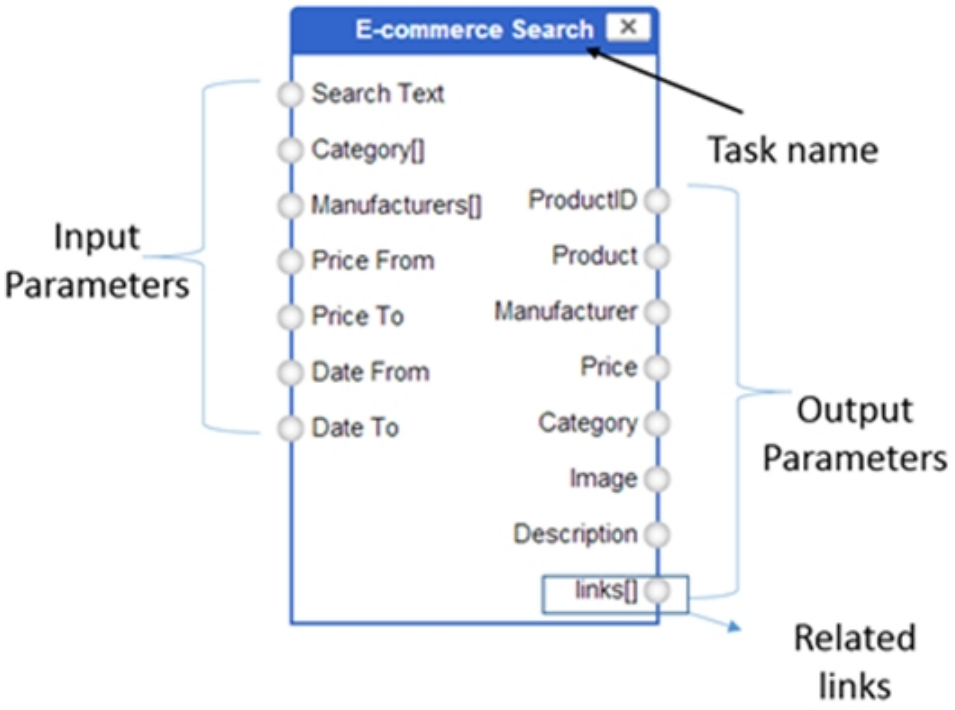


Figure 3.17: Extracted resource from a web application

3.5.5 Prototype for Identifying RESTful Resources from Web Applications

Each extracted task as a RESTful resource is represented in Figure 3.18. Each resource has a resource name, input parameters and output parameters and links to related resources.

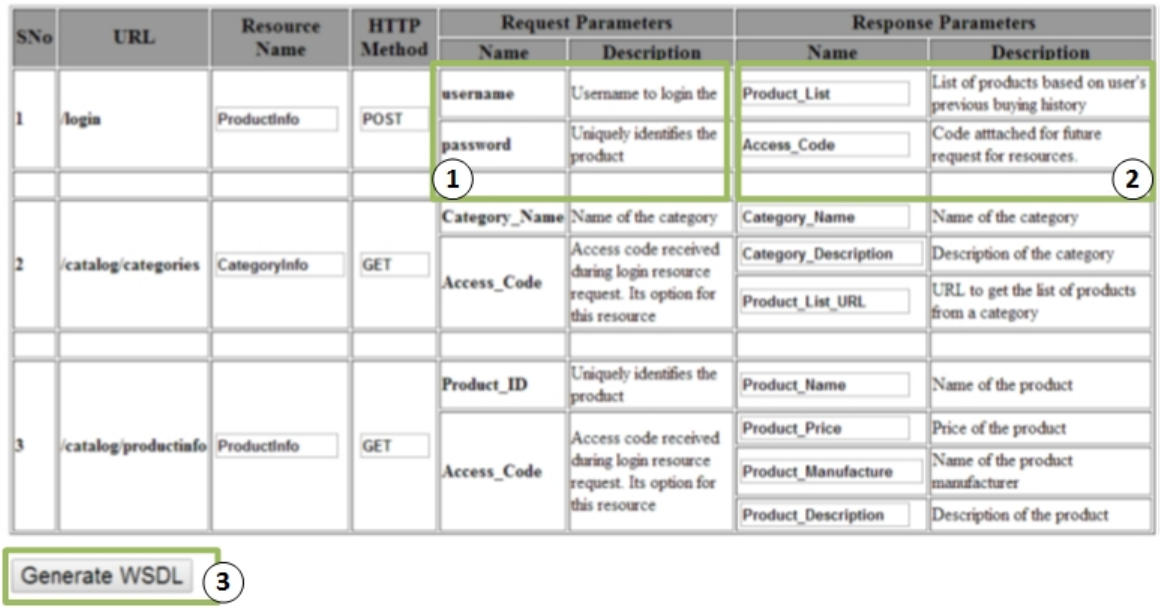


Figure 3.18: GUI with identified resource from a web application

We have designed and developed a prototype of our proposed approach to migrate web applications to RESTful services. We provide a tool (as a Firefox plugin) that helps a user to identify the task, and annotate the resource representation. Figure 3.13a shows annotated screen shot showing the initial and final state of the task. Figure 3.13b helps a user to annotate the resource representation. A user can verify all the identified resources as shown in Figure 3.18. Figure 3.18 shows the resource names, the inputs required for a resource and the output of a resource.

3.6 Case Studies

In this section, we discuss case studies to identify resources from SOAP-based services and web applications. Our initial survey of 713 service descriptions, as in Table 3.4, shows the existence of both SOAP-based services and RESTful services. Based on our observation while collecting services from the Internet, we found that RESTful

Table 3.4: Different types of services

Domain	RESTful service	SOAP-based service
Internet	160	55
Shopping	97	17
Search	84	11
Financial	64	57
Enterprise	74	29
Government	61	4
Total	540	173

services are more popular than SOAP-based services. Especially, RESTful services are popular in data-oriented services. In most of the RESTful services, the developer network is maintained in the form of blogs, social networks, and forums engaging the developers around the world.

To identify RESTful services from web applications, we build a tool (as a Firefox plugin) that helps a user to identify tasks, and annotate the resource representations. Figure 3.13a shows annotated screen-shot showing the initial and final state of a "login" task. Figure 3.13b demonstrates our tool to annotate the resource representation. A user can verify all identified resources for a task as shown in Figure 3.18. We provide a GUI to verify resource extraction. A user verifies all the request, and response parameters and then generates a service description document. In this case study, we evaluate our approach to extract RESTful service from web applications. We discuss our case study to evaluate the effectiveness of our proposed approach to extract tasks, and task relations from a web application. To ease the verification of our results, we selected web applications that already have exposed web services. We consider each resource as a task and compare the effectiveness of our approach. More specifically, we aim to assess (1) the effectiveness of our approach to extract task from a web application with correct input parameters and output parameters, and (2) the

Table 3.5: Identify RESTful services from SOAP-based services

Resource	WSDL methods	HTTP methods
addCountry	/{country}	POST
getAllCountries	/{country}	GET
deleteCountry	/{country}	DELETE
updateCountry	/{country}	PUT
addCity	/{country}/city	POST
getCities	/{country}/city	GET
updateCity	/{country}/city	PUT
deleteCity	/{country}/city	DELETE
getCapital	/{country}/capital	GET
addCapital	/{country}/capital	POST
updateCapital	/{country}/capital	PUT
deleteCapital	/{country}/capital	DELETE

effectiveness of our approach to identify task relations. In the following sub-sections, we discuss the case study in more detail.

3.6.1 Setup

To test the performance of our REST Wrapper for the SOAP-based services, we first used our approach to identify RESTful services. Table 3.5 shows the result of the identified resources, HTTP methods and the WSDL operations of a SOAP-based service. We generated the necessary wrappers and configuration files to deploy the RESTful service. Both RESTful and SOAP-based services can be invoked. We deployed both services on the same computer. We developed a client application for each service. Client applications are executed in a separate network than where services are hosted. Client applications invoke services a fixed number of times.

Table 3.6: Domain and number of SOAP-based services used in the case study

Domain	#	Description
Finance	12	Services related to financial management and banks
Government	6	Services provided by government organizations
Travel/Tourism	17	Services that are related to travel and tourism e.g., flight book, hotel booking, and taxi reservation
E-commerce	13	Services provided by online business e.g., Amazon, BestBuy and EBay
Others	13	Services from domains such as weather, music search, content sharing and aggregation

To validate our approach for identifying the resources from SOAP-based web services, we collect 61 WSDL documents from various categories, such as finance, government, travel/tourism, and e-commerce. The services are downloaded from online sources such as WebserviceX [131], WebserviceList [130], and xMethods [141]. Table 3.6 list the categories, the number of WSDL document in each category and a short description of the category. Our classification of categories is based on an online web service listing site [100]. We identified resources and HTTP methods from each WSDL as discussed in Section 3-2. We analyzed the WSDL documents manually to examine the accuracy of the identified resources and the corresponding HTTP methods.

We randomly choose 21 web applications from five different domains (*i.e.*, finance, weather, e-commerce, book and travel). We make sure there are services for these web applications. Choosing web applications that have already exposed services helped us to use services as tasks for the case study. Table 3.7 lists the domains, and the number of web applications selected to assess our approach.

The computer used for case study has Intel Core 2 Duo of 2.66 GHz, and RAM

Table 3.7: Domain and number of web applications used in the case study

Domain	#	Example web application
Finance	4	http://www.exchangerate.com
Weather	3	http://www.theweathernetwork.com
E-commerce	6	http://ebay.com
Book	5	http://www.freebooksearch.net/
Travel	3	http://www.expedia.ca/Flights

4 GB. The server is an Apache Tomcat server of version 6.0.29 and AXIS of version 1.3.

3.6.2 Evaluation Criteria

We measure the effectiveness of our approach on identifying resources using precision and recall. Precision can be seen as a measure of exactness or fidelity [76] as shown in Equation 3.2. Precision measures if any irrelevant resources are misidentified as the resource. Recall is a measure of completeness [76] as shown in Equation 3.3. Recall evaluates whether our approach can correctly identify all resources without omissions. We could not compare the precision and recall with other works because based on our knowledge, there is not publicly available experimental data in migrating SOAP-based services to RESTful services.

$$precision = \frac{|\{Ri\} \cap \{Rj\}|}{|\{Ri\}|} \tag{3.2}$$

$$recall = \frac{|\{Ri\} \cap \{Rj\}|}{|\{Rj\}|} \tag{3.3}$$

where $\{Ri\}$ and $\{Rj\}$ are:

Case 1: For evaluating the effectiveness of identifying resources from

SOAP-based services.

$\{Ri\}$ *is the number of identified RESTful services by our approach.*

$\{Rj\}$ *is the total number of RESTful services by manually examining SOAP-based service descriptions.*

$\{Ri\} \cap \{Rj\}$ *is the number of correctly identified RESTful services from SOAP-based service descriptions.*

Case 2: For evaluating the effectiveness of identifying task relations.

$\{Ri\}$ *is the number of identified task relations by our approach.*

$\{Rj\}$ *is the total number of task relations by manually examining the tasks.*

$\{Ri\} \cap \{Rj\}$ *is the number of correctly identified task relations in a web application.*

Case 3: For evaluating the effectiveness of identifying the input and output for a task.

$\{Ri\}$ *is the number of input and output for a task identified by our approach.*

$\{Rj\}$ *is the total number of input and output required by the task.*

$\{Ri\} \cap \{Rj\}$ *is the number of input and output for a task correctly identified. The correctness includes correctly identified semantic labels of input and output.*

For the performance between SOAP-based services and RESTful services, we measure the time difference between sending a request and receiving the response from the server for both types of services. Performance is the ratio of difference between response time taken by SOAP-based services and RESTful services by the response time taken by the SOAP-based services. Equation 3.4 gives the performance measure.

$$Performance = \frac{SOAP_{Average_{time}} - REST_{Average_{time}}}{SOAP_{Average_{time}}} \tag{3.4}$$

where $SOAP_{Average_{time}}$ *is an average response time for SOAP operation and*

$REST_{Average_{time}}$ is an average response time for RESTful service.

We inspect the results produced from our approaches. The inspector has two years of experience in web APIs and website development. For the case studies, the inspector manually checked if the proposed approach can correctly identify tasks, task relations and input and output of the task.

3.6.3 Analysis of Results

In this sub-section, we discuss our findings during RESTful service extraction process. The sub-section is further divided into two parts discussing our finding to identify RESTful services from SOAP-based services and web applications.

Analysis for Identifying Resources from Web Applications

We found the execution of the RESTful service is a slower than loading the original web pages in a web application. It is because of data extraction is done as a background job the information takes some time. Data transfer between the client and the server is lower as only the specific data is sent to the client. This is because the client only issues a single HTTP-interaction to complete a task and it gets only the meaningful content as output. This low bandwidth consumption makes our approach suitable to make applications for bandwidth and battery critical devices such as tablets and mobiles.

To identify if our approach can correctly extract tasks, we choose a number of tasks from different domains as listed in Table 3.7. Table 3.8 lists the average number of tasks extracted by our approach. We find the extracted resources containing all the three kinds of resources (*i.e.*, Type I, Type II and Type III). Table 3.8 lists the

Table 3.8: Result of identifying Input/Output for Tasks

Domain	**# Average task Identified**	**# Average input & output**	**Precision of Input / Output**	**Recall Input / Output**
Finance	10	24	80%	93%
Weather	3	6	100%	100%
E-commerce	12	58	78%	95%
Book	8	74	81%	92%
Travel	12	82	79%	96%

Table 3.9: Result of identifying task relations from web applications

Domain	**# Identified task relations**	**Precision**	**Recall**
Finance	7	85%	100%
Weather	2	100%	100%
E-commerce	10	78%	100%
Book	8	77%	100%
Travel	7	82%	100%

results showing the effectiveness of identifying the input/output parameters and their description (*i.e.*, labels) for a task. Our approach achieved a satisfactory result (i.e., above 80% precision) on identifying the input/output for a task. The high recall (*i.e.*, above 95%) shows our approach recovers most of the input/output parameters.

To evaluate if our approach can correctly identify resource relations from the task listed in Table 3.8. Figure 3.19 shows the resource relations identified among different task in web application from e-commerce domain. Table 3.9 lists the result showing the effectiveness of identifying the tasks relations. Our approach has the satisfactory result (*i.e.*, above 80% precision) on identifying the resource relations. The high recall (*i.e.*, 100%) shows that our approach finds all the resource relations.

Our approach misidentified some of the input and output element labels because of the complex layout and nested structures of web forms and the response pages.

Some of the input elements have default values without having any descriptive text. Similarly, the use of graphic images instead of text hinders the identification of the description of an element.

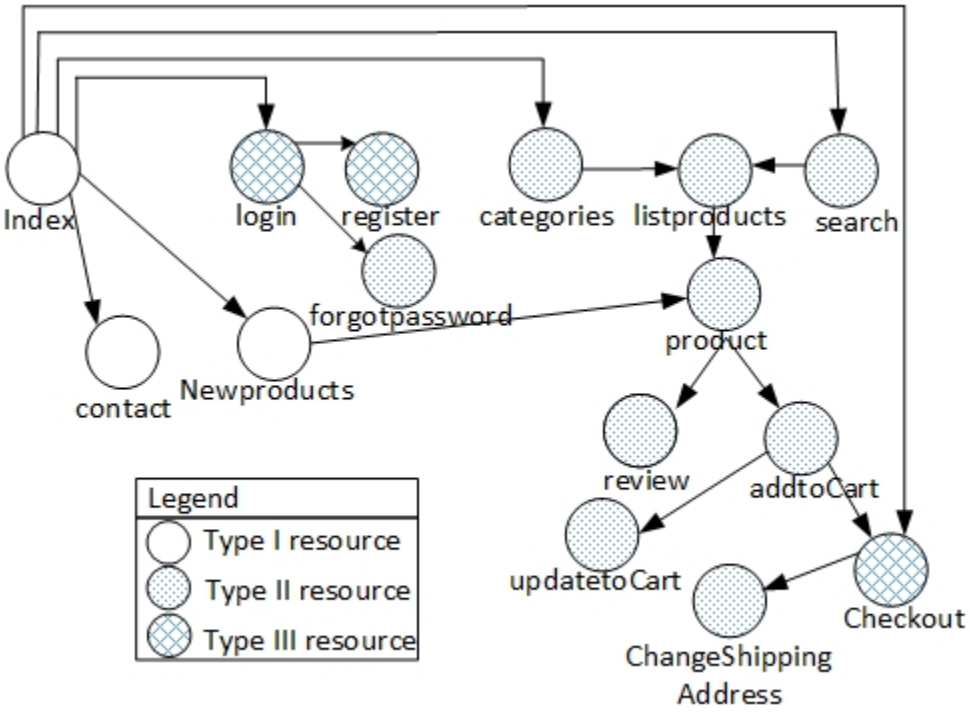


Figure 3.19: Task relation between tasks extracted from a web application in the E-commerce domain

Figure 3.19 shows the task relations identified among different tasks a web application in the e-commerce domain. Table 3.9 lists the result showing the effectiveness of our approach in identifying task relations. Indeed, our approach has a satisfactory result (*i.e.*, above 80% precision) on identifying resource relations. The high recall (i.e., 100%) shows that our approach can recover all resource relations.

Using our approach, a developer extracts services from the web applications that have not exposed web services and the web applications whose source code is not

available.

Analysis for Identifying Resources from SOAP-based Services

To validate the effectiveness of our approach, we conduct a case study to 1) assess the accuracy of our approach of identifying resources from existing SOAP-based services; and 2) evaluate if the migrated RESTful services can fully use the benefit of the REST architecture. We compare the benefits in terms of response of the migrated RESTful services and the response time of the SOAP-based services.

Table 3.10: Results of identifying resources from WSDL

Category	**# Methods**	**# Predicted Resources**	**# Misidentified Resources**	**# Total Resources**	**Precision**	**Recall**
Finance	59	35	8	40	77%	67%
Government	39	25	2	27	92%	85%
Travel/Tourism	132	97	16	110	83%	73%
E-commerce	102	80	14	90	82%	73%
Others	78	87	6	53	87%	77%

Table 3.11: Summary of WSDL to RESTful service identification

Number of WSDL documents Analyzed	61
Total Number of Operations	410
Number of Resources Identified	284
Misidentified Resources	46
Total Number of Resources	320

Table 3.10 lists the result for identifying resources from SOAP-based services. The average precision is above 84%, meaning that our approach can correctly identify the services most of the times. The recall of our approach is 75%. The recall shows that our approach may fail to identify the resources correctly. We were not able to identify the resource correctly when the names of the input and output parameters are generic. Due to unclear name in the operation, tunneling the operation name through POST operation was identified. Use of ambiguous words and words not available in WordNet causes difficulties in identifying the resources correctly. Table 3.11 summarizes the results of accuracy of the identified resources. The accuracy is 74% showing that our approach can successfully identify the resources in most of the cases. There are 410 operations in those 61 WSDL documents, for which there were around 284 resources. Therefore, each resource may not always have four operations exposed for the users. We found most of the web services in our study to be read-centric as most of the resources exposes GET method.

Table 3.12 shows the result of service response time for both SOAP-based and RESTful services. The performance improvement is calculated using Equation 3.4. Even with the delay introduced by the wrapper, the performance of RESTful services is better than of SOAP-based services.

Table 3.12: Performance of using WSDL client and RESTful version of the same WSDL service

SOAP-based Service	**Identified RESTful Service**		**SOAP-based Service Average Response Time (ms)**	**RESTful Service Average Response Time (ms)**	**Performance Improvement in Response Time**
Operation Name	**Resource**	**HTTP Method**			
addCity	/{country}/city	POST	78	52	33%
getCitiesByCountry	/{country}/cities	GET	79	49	37%
deleteCountry	/country	DELETE	80	46	42%
updateCountry	/country	PUT	85	65	23%

3.6.4 Threats to Validity

The main threat to our experiments that could affect the generalization of the presented results relates to the number of WSDL documents and Web applications analyzed. We have analyzed 61 SOAP-based services and 21 web applications from a wide variety of industries. Nevertheless, further validation of our approach requires analyzing a larger set of WSDL documents and web applications. A service provider may have multiple interdependent WSDL documents; analysis of multiple files together may affect the resource identification results.

There are different ways to denote a resource. We make resources more readable, following the hierarchical rule in NLP, but resources can be denoted in other ways as well. The network congestion and traffic are not taken into account for the performance evaluation of SOAP-based and RESTful service invocation.

3.7 Summary

In this chapter, we describe a model to represent heterogeneous services in a unified format. We propose a semi-automatic technique to identify resources from SOAP-based services and web applications. We use cluster analysis and natural language processing to identify the resource names and the associated HTTP methods. The results from the case studies show that our approach can identify the resources with high precision. The case studies also show that RESTful services have performance benefits compared to SOAP-based services. Similarly, our approach reduces bandwidth consumption to perform a task through RESTful service invocations rather than through web applications. We analyze client side user interfaces and fragments of the output representation to extract reusable tasks. Our approach currently does

not support extraction from Silverlight [110] or Flash [43]. Our approach uses the Firefox JavaScript Engine [112]. Hence, codes specifically developed to be executed in some other browsers (*e.g.*, Internet Explorer, Opera, and Safari) are not extracted.

Chapter 4

Concept-based Service Discovery

Web services are the basic constructs to build complex distributed applications with fast speed and low cost. However, the existing service discovery techniques provide the users with poor results which require substantial human intervention to filter the services to locate the desired ones. The queries formulated by the users may not match well with the service descriptions of the existing services. As a consequence, a user's query can result in a large number of returned services. In this chapter, we propose an approach that derives the semantic concepts conveyed in the service descriptions and clusters the services based on the derived concepts. As a result, each concept is associated with a set of relevant services. To understand the semantic meanings of a user's query, we identify the concepts behind the query and recommend the related concepts associated with the services. Our approach guides the users to formulate their queries.

4.1 Motivation

With the ever-increasing number of services published on the Internet (*e.g.*, Google [49] has indexed 204,000 WSDL documents), it becomes challenging to find the desired

services. Therefore, an effective service discovery mechanism is essential to help the developers harness the benefits offered by the web services. The large body of research on service discovery can be summarized into three main categories: semantic web approaches [58, 61, 114]; web search engines [15, 49, 107]; and information retrieval (IR) approaches [4, 37, 71]. Semantic web approaches propose to enhance the service descriptions, instead of exploiting them as they are used as new tools for discovering web services. Service descriptions usually reside in web servers. Web search engines crawl and index the contents of the servers and enable the users to retrieve desired web services. IR techniques, such as word sense disambiguation, stop-words removal, and stemming are used to extract relevant information conveyed within the service description documents to index web services. However, the existing approaches in service discovery suffer from the following limitations:

Gap in the semantics of the service descriptions and the search queries. The precision of the service discovery approaches is dependent on the understanding of the semantics of the service description documents and a user's query. The vocabulary adopted in the service description is often used by the developers in the software development domain. It can be very different from the ones specified in the web service search queries. For example, a user may query for transportation service; while the service providers may represent specific types such as car, taxi, and bus. Without a good understanding of the semantics of the service description and the search query, the service discovery system is likely to retrieve a large number of irrelevant web services or to have no return.

Lack of support for query formulation. The available service descriptions act as a black box to the users. A user has to conduct many trials to formulate an

appropriate query to retrieve the desired services when an initial query fails. A user cannot specify the searching criteria based on a particular requirement. For example, to reserve a two bed hotel room in downtown Toronto, users may either write a whole sentence or specify a few keywords, like "reserve room". In both cases, users may receive too many results that need a considerable amount of manual filtering. Therefore, it is critical to provide the users with efficient ways to articulate the service queries (*i.e.* considering the inputs and the outputs of the operations) to improve the precision of the service discovery.

To address the aforementioned limitations, we propose an approach that identifies the semantic meanings of service descriptions and users' queries as concepts. A concept is a semantic notion or a keyword for describing a subject, *e.g.*, "traveling", "weather" or "taxi reservation". In particular, we identify a set of concepts from the service description documents. We further index the services in terms of their associated concepts and the relations among concepts using WordNet [86], a lexical database. Moreover, our approach assists the users in formulating their queries using similar vocabularies to the ones specified in the service description documents. We use the common-sense knowledge (*e.g.*, checking reviews before watching a movie), encoded in ConceptNet [72] to link the concepts delivered in the service descriptions and the concepts extracted from a user's query. We provide a graphic tool that guides the users to select the generated concepts, and allows them to combine different concepts to better describe their searching criteria.

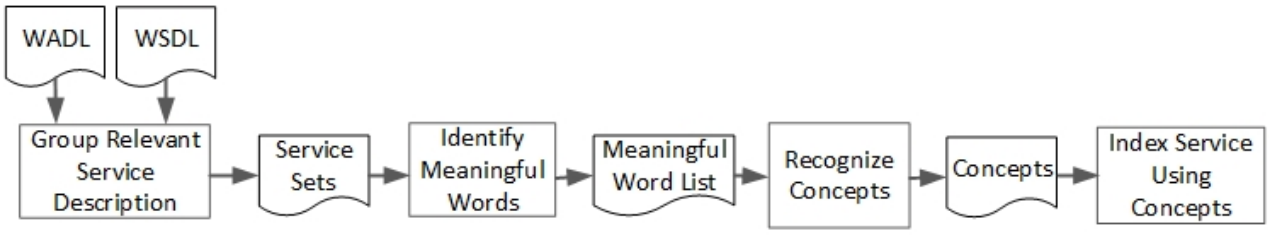


Figure 4.1: Overall steps for indexing services using the concepts extracted from service description documents

4.2 Overview of Our Approach

Our approach consists of two major steps: 1) service grouping based on the concept shared between the service sets; and 2) service retrieval which returns service based on the user's query. Service grouping is an offline step that extracts the concepts from the service description documents. The grouping is done based on the shared concept between the services. The service retrieval component is an online step that extracts the concepts from a user's query and with the obtained concepts it retrieves related services from the service repository. In the following sub-sections, we discuss the two steps in more details.

4.2.1 Service Indexing

Figure 4.1 gives an overview of the steps involved in the service indexing process. We explain each step as follows:

Grouping Relevant Service Descriptions

A service can be an operation in a SOAP-based service or a resource in a RESTful service. However, such descriptions are often scattered throughout a web service

```
<portType name="CurrencyExchangePortType">
        <operation name="getRate">
        <input message="tns:getRateRequest">
          <message name="getRateResponse">
                <part name="country1" type="xsd:string />
                <part name="country1" type="xsd:string />
          </message>
        </input>
        <output message="tns:getRateRequest"/>
          <message name="getRateResponse">
                <part name="Result" type="xsd:float"/>
          </message>
        </output>
        </operation>
</portType>
```

Figure 4.2: Rearranged service description of a WSDL

description document. For example, in Figure 4.2, XML fragments related to "getRate" is scattered in multiple elements, such as "portType" and message in the description document. Instead of analyzing the entire service description document, we rearrange tags in the service description documents and assemble the information related to a service (*i.e.* an operation or a resource) in one location. The information related to a service is called a service description set. A service description document can have multiple service description sets due to multiple operations or resources. For SOAP-based services, the service description set contains an operation, its input parameters, its output parameters, and the documentation corresponding to the operation. Similarly, for RESTful services, the service description set includes a resource name, HTTP method, the URL of the resource, request/response parameters, and the documentation.

Identifying Meaningful Words

To identify concepts from each service description set, we first extract the name of the service (*i.e.* operation or resource), the messages and the parameters in a service description set. The extracted names can be compound words. For example, two words are joined by the change of case (*e.g.*, "findCity"); words are separated by underscore (*i.e.* "_") or hyphen (*i.e.* "-") (*e.g.*, "find_city"), and words are added with a suffix (*e.g.*, "city1"). We tokenize the compound words into single words. We use WordNet to filter valid English words.

Some names used in the service descriptions are not separated by special symbols (*e.g.*, "_", "-" or camel case). For example, a RESTful service describes a resource using a URL. The names are not valid English words either. To identify the meaningful words, we perform an n-gram analysis, which can be used to detect the sub-words within a word by extracting a contiguous sequence of n letters from a string. For example, the 3-gram analysis of a word, "improve", extracts all the possible consecutive three letters in the string, *i.e.* "imp", "pro", "rov", "ove". We perform the n-gram analysis starting from 3-grams (*i.e.* i-grams and i=3) to check if the three consecutive letters in a string are a valid English word using WordNet. Then, we iteratively increase from i-grams to i+1- grams to recognize valid words and continue until i+1 is the maximum length of the string. For example, the n-gram analysis of the URL, http://bookmooch.com/topic/{topic}, generates a vector, *i.e.* {book, topic, mooch}.

Moreover, we reduce the words to the root words using the Porter stemmer [99]. For example, 'reserve', 'reserved', 'reserving', and 'reserves' have the same stem 'reserve'. All these words have similar semantic meanings. We use the rules listed in Table 3.1 (in Chapter 3) to decompose the words to identify meaningful words. Each

service description set has one or more words extracted from the service (*i.e.* operation or resource), the inputs and the outputs. We classify the words into three groups: service (*i.e.* resource and operation) Keyword (SK)); input parameters (*i.e.* IP); and output parameters (*i.e.* OP). The extracted words are stored in the format shown in Equation 4.1.

$$SWord = \{(Wi, freq, type)\} \tag{4.1}$$

where SWord is a list of words with frequency and type; Wi is an element in the word list, SWord; freq is the frequency of Wi; and type $\in$ { *SK, IP, and OP* }.

A word list (*i.e.* SWord) stores each word (*i.e.* Wi) extracted from a service description set, the frequency (*i.e.* freq) of the word (Wi) appeared in a service description set. The type attribute separate the SWord into three groups (*i.e.* SK, IP and OP).

Recognizing Concepts

A word list (*i.e.* SWord) may contain a lot of words. However, not all the words reflect the functionalities of a service. In particular, the names used in a service often consist of the verb and noun, such as getCustomer and deleteCustomer. As a result of the last step (*i.e.* extracting meaningful words), get, delete and customers are valid words in the word list. However, the general operational words, such as delete, update and get, describe the manipulation on the data without indication of the functionality of a service which is related to customer services in the example. Our aim is to identify and filter general operational words from the words extracted from a service description set. We consider the remaining words as functional words,

i.e. concepts.

To separate words in the word list, SWord, into two clusters (*i.e.* general operational words and functional specific words). We apply k-means algorithm [54], a clustering algorithm, on the word list. We set the number of clusters generated to 2 (*i.e.* k=2). K-means algorithm is selected due to its simplicity and efficiency. The clustering algorithm groups similar words using the semantic similarity as shown in Equation 4.2.

$$wordSim(x, y) = 1 - \frac{mcp(cp)}{mcp(cp) + dcp(cp, root)} \tag{4.2}$$

where cp is the common parent of the two words x, y; root is the root of the WordNet ontology; mcp(cp) is the shortest path from either x or y to cp; and dcp(cp, root) is the length of the path from cp to root.

Concepts in WordNet can be connected with different types of relations such as hypernym, hyponym and holonym. The two concepts can be directly or indirectly connected through many intermediate relations and concepts. A path length is the number of intermediate concepts to traverse from one concept to another. The similarity between two concepts x and y is measured by the path length between the concepts to reach their common parent in the WordNet ontology. The value of the similarity metric shown in Equation 4.2 ranges from 0 to 1. 0 represents unrelated words and 1 signifies synonymous words.

To determine the cluster that contains the general operational words, we pre-define an oracle of general operational words (*e.g.*, "update", "post", "add", and "create") by manually examining a number of service description sets in different domains. The cluster semantically closer to this oracle is determined to contain the operational

words and discarded from the word list (*i.e.* SWord). For example, the operation "getRate" shown in Figure 4.2, we extract a set of words from the service description, *i.e.* {get, country, currency, rate, exchange, request, result}. We further divide the words into two clusters, *i.e.* {get, request, result} and {country, rate, exchange, currency}. The cluster {get request, result} is close to the predefined oracle and hence is discarded. The remaining cluster {country, rate, exchange, currency} contains the functional words, *i.e.* the concepts representing the functionality of the service.

Indexing Services using Concepts

There are more than one concepts describing the functionality of a service. Some of the concepts might be redundant. Other concepts can be less frequently used. We identify a set of representative concepts that can capture the major functionality delivered by a service. We consider the most frequently used concepts as the representatives of the functionality. As aforementioned, the concepts in the refined word list (*i.e.* SWord) are divided into three groups (*i.e.* SK, IP and OP). For example, the concept set {country, rate, exchange, currency} contains {exchange, rate, currency} associated with the operation (*i.e.* SK) and {country} derived from the input (*i.e.* IP). We treat each group individually and rank the frequency of a concept based on the semantic similarity among the concepts within the group. Equation 4.3 computes the rank of concepts in each group.

$$R(x) = \sum_{y \in C; y \neq x} WordSim(x, y) f(y) + f(x) \tag{4.3}$$

where $R(x)$ denotes the rank of the concept x in the cluster C; $WordSim(x, y)$ is the similarity between concept x and y; and $f(x)$ is the frequency of the concept x.

The rank of a concept x (*i.e.* R(x)) is the sum of the frequency of concept x and the frequency of concept y prorated by the semantic similarity between the concepts x and y. The rank of concept x increases if the cluster has more concepts semantically similar to concept x. Ranking the concepts signifies the frequency of a concept with respect to other concepts in a group. The computed rank is then normalized between 0 to 1 by dividing a rank of a concept with the sum of all concept ranks in a group. 1 signifies the most dominant concept. For example, the similarity between "exchange" and "currency" is 0.5; "exchange" and "rate" is 0.3; and "rate" and "currency" is 0.26. Using these similarity values, we compute the rank of the concepts {exchange, currency, and rate}, the concept rank as described in Equation 4.3 is {exchange (0.5+0.3+1=1.8), currency (0.5+0.26+1=1.76), rate (0.26+0.3+1=1.56)}. If we select two representative concepts, it will be exchange and currency. The input has only one concept and its rank is {country (1)}. The number of representative concepts to use is a design decision. We manually examined a few service description sets and found that the two concepts can effectively represent a group. Thus, we use two representative concepts for each group. The operation "getRate" in Figure 4.2 is indexed under the concepts "currency" and "exchange" due to the semantic relation with the operation concept and also associated with concept country because of the relation with the input concept.

4.2.2 Service Retrieval

A user expresses his queries using natural language. Studies on the users' web search behaviors have shown that a large portion of the web search queries consist of one to three keywords [113]. Short queries indicate that the users often have difficulties

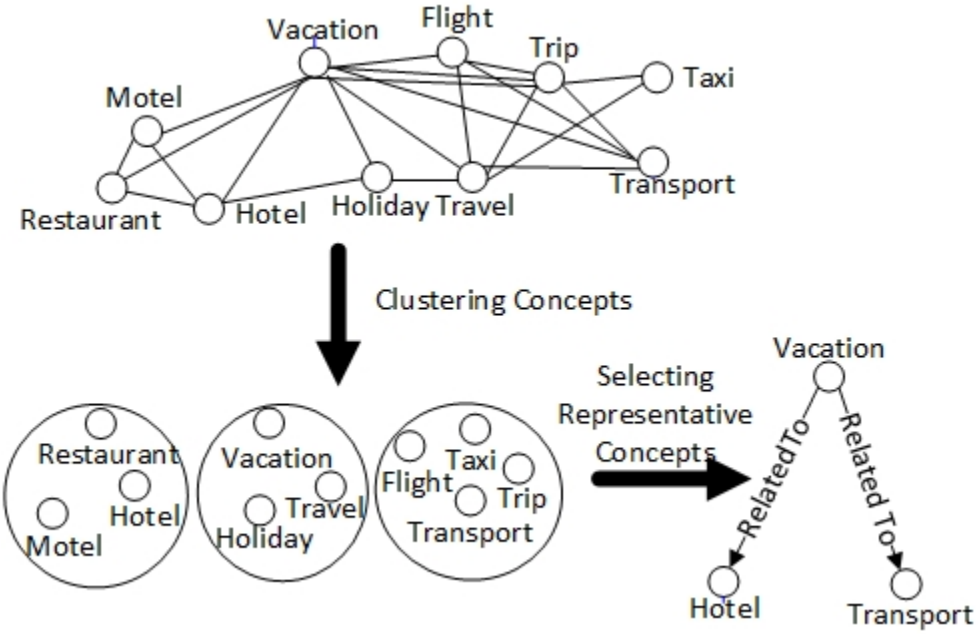


Figure 4.3: Process for creating a concept map

in crafting queries to reflect the information needed. To understand users' intention and requirement, we extract concepts from the users' queries and retrieve semantically related concepts from the service sets (SWord). Users can select one or more concepts from the retrieved concepts from the service sets to formulate their query. This can provide some knowledge on the available services and help users articulate their specific needs. Hence we can provide services with the best matches. The service retrieval consists of two steps: 1) analyzing queries; and 2) grouping services.

Analyzing Users' Query

A user query can have two characteristics: 1) a few keywords containing key concepts; and 2) a long verbose query including "wh-" questions (who, what, when, where). In the former case, the concepts are directly provided by the user. In the latter case,

we parse the user's query to analyze the structure of the sentences in the query and identify the concepts from the sentences.

We create a concept map to help a user visually select concepts to formulate their query. A concept map is a set of concepts identified from a user's query and their relations. The concepts are represented as nodes. A relation between two concepts is derived from ConceptNet and illustrated as an edge linking the concepts. To create a concept map, we cross-reference concepts in a query with semantically related concepts in the service repository. For example, if the concept "vacation" is present in a user's query, we retrieve concepts related to "vacation" from ConceptNet, provided that there are services associated with the retrieved concepts. For example the concept "vacation" is related to {motel, hotel, restaurant, flight, trip, taxi, transport, travel, holiday} as shown in Figure 4.3. There may be a large number of related concepts which can be overwhelming to users. We cluster related concepts by measuring their similarity as defined in Equation 4.2. Each cluster is represented by a representative concept.

$$Rep(x) = \sum_{y \in C; y \neq x} WordSim(x, y) \tag{4.4}$$

where $Rep(x)$ is the rank of the concept in a cluster; and $WordSim(x, y)$ is the similarity between concepts x and y.

Equation 4.4 computes the rank of each concept with respect to other concepts in the cluster. A concept with the highest rank is considered as the representative candidate of the concept cluster. We represent the selected representative concepts and their relations as a concept map. Figure 4.3 shows the process of creating a concept map. Table 4.1 shows an example of concept clustering and selection of a

Table 4.1: Examples of concept clusters and representative concepts

Concept in the query	Cluster of related concepts	Representative concepts
Holiday	{car, taxi, vehicle} {hotel, motel, hostel}	{vehicle} {hotel}
	{airplane, flight}	{flight}
Hotel	{car, taxi, vehicle}	{vehicle}
	{motel}	{motel}
	{hostel}	{hostel}
Weather	{climate}	{climate}
	{wind}	{wind}
	{temperature}	{temperature}
Flight	{aeroplane, plane}	{aeroplane}
	{car, taxi, vehicle}	{vehicle}
	{holiday}	{holiday}
Computer	{monitor}	{monitor}
	{CPU, hard disk}	{CPU}
	{keyboard, mouse}	{keyboard}
Camera	{review}	{review}
	{feedback}	{feedback}
	{ranking, rank}	{rank}
Entertainment	{movie, theater, film}	{movie}
	{casino, sport}	{sport}
	{nightlife, dance}	{nightlife}

representative concept. For simplicity, we use number of clusters as 3. A concept map should not contain a lot of recommendations that requires a considerable time for a user to explore. Miller [85] claims that there are 7 ± 2 slots available in human short-term memory. To guide a user to make an instant decision, we list maximum seven concepts and their relations in a concept map.

To allow a user to conduct a specific search in terms of the types of services (*i.e.* operation in SOAP-based web services or resource in RESTful services), the input and the output, we define a simple query syntax for a user to specify the location of a concept (*i.e.* input, output and operation) in the query and combine more than

one concepts using operators, such as, "and", "or", and "exclude". For example, a search for a service that returns hotels by postal code can be specified as "postal code ctype:IC & hotel ctype:IC", meaning that the user wants to find services with the hotel concept delivered in a service and the concept postal code as input.

Grouping Services

The returned result may contain multiple services. Grouping the returned services makes it easier for a user to find a specific service. Web services can share common attributes, such as the location of services, Quality of Service (QoS) and the number of shared common concepts. However, most of the services do not have QoS information. In our work, we group services using the common concepts between operations or resources (SK), inputs (IP) and outputs (OP). More specifically, our approach groups the retrieved result into four categories with varying restrictive levels: 1) services that share common concepts in SK, IP and OP; 2) services that have common concepts in SK and OP; 3) services that contain common concepts in SK and IP; and 4) services that share common concepts in SK only. Each group consists of the similar type of service description sets.

$$C(x,y) = \begin{cases} \frac{|IP(x)\cap IP(y)|\times|OP(x)\cap OP(y)|}{|IP(x)\cup IP(y)|\times|OP(x)\cup OP(y)|} & \text{if } IP(x)\cup IP(y)\neq\phi \wedge OP(x)\cup OP(y)\neq\phi \\ \frac{|IP(x)\cap IP(y)|}{|IP(x)\cup IP(y)|} & \text{if } IP(x)\cup IP(y)\neq\phi \wedge OP(x)\cup OP(y)=\phi \\ \frac{|OP(x)\cap OP(y)|}{|OP(x)\cup OP(y)|} & \text{if } IP(x)\cup IP(y)=\phi \wedge OP(x)\cup OP(y)\neq\phi \\ 0 & \text{if } IP(x)\cup IP(y)=\phi \wedge OP(x)\cup OP(y)=\phi \end{cases} \tag{4.5}$$

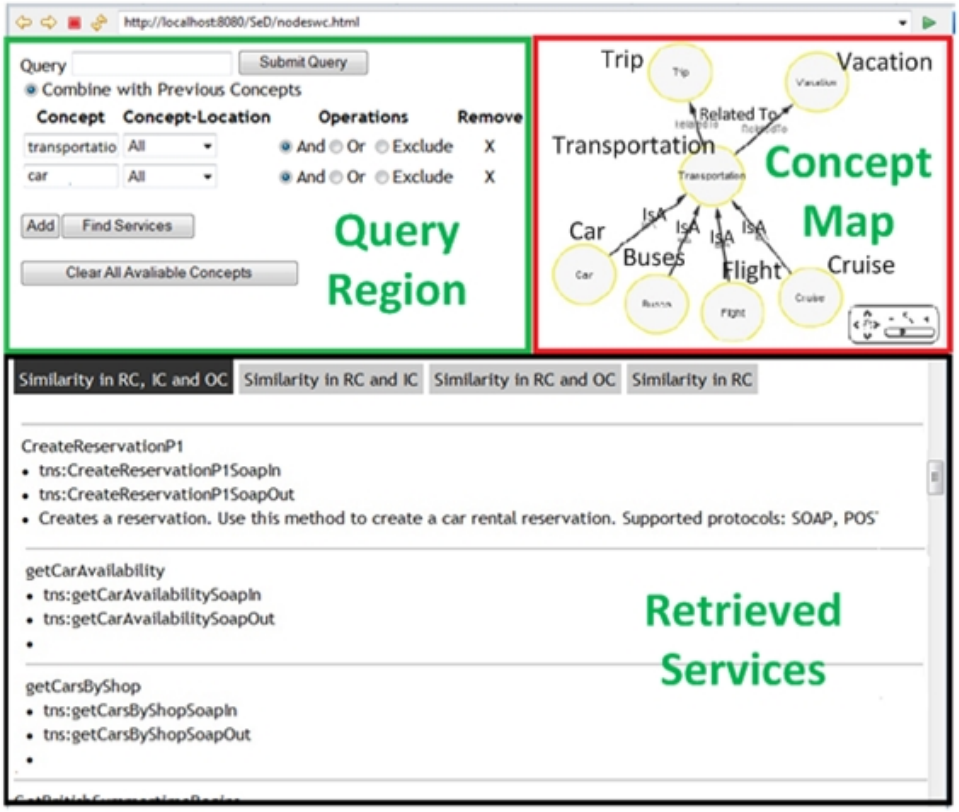


Figure 4.4: Annotated screen-shot of our prototype

where $C(x, y)$ is the ratio of common concepts to the total concepts between service description set x and y; $IP(x)$ is the Input concept of the service description set and $OP(x)$ is the output concept of the service description set x

Equation 4.5 measures the similarity between the two services. It is the ratio of the common concepts divided by the total number of concepts of the two services. The services in each group are ranked based on the sum of the rank of a concept in services and the rank of a query concept in the service description sets. We compute the rank of a concept extracted a query using Equation 4.3.

4.3 Prototype for Concept based Service Discovery

We develop a prototype of our proposed approach. Figure 4.4 shows the annotated screen-shot of the user interface (UI) of our prototype. The user interface is divided into three areas: "Query Region" where a user can submit a query and select concepts and concept locations (*i.e.* input, output, or a service name); "Concept Map" shows the concepts extracted from a query as well as the related concepts to the query using ConceptNet; and the "Retrieved Service" area lists the retrieved services.

As shown in Figure 4.4, a user submits a query (*i.e.* "transportation in Kingston") in the "Query Region"; our prototype retrieves the related concepts shown in the "Concept Map" area. The "Concept Map" displays different types of transportation, such as car, flight and cruise. These concepts serve as an initial point to help a user to craft a more specific query. Once a user finds a relevant concept in the "Concept Map", the prototype retrieves services related to the concept and lists them in the "Retrieved Services" area. To ease the user to go through the services, our prototype further divides the services into four categories from the strict matching of the concepts appearing in all three fields (*i.e.* the input, the output and the name of a service) to a looser matching which matches one concept in one of the fields (*i.e.* input, output or service name).

4.4 Case Studies

We conduct a case study to evaluate the effectiveness of our approach. The objectives of the case study include: 1) evaluate the effectiveness of our approach to index services based on the concepts extracted from service description documents. We compared the precision and the recall of our approach with a baseline approach which

Table 4.2: Descriptive statistics of our data set

# WSDL documents for SOAP-based web Services	550
# WADL documents for RESTful Services	61
# Service Description Set (i.e., Services)	10100
# Concept Identified	4100

indexes services using textual description (*i.e.* service documentation) of web services; and 2) examining the effectiveness of our approach for concept recommendation and query formulation through a user study.

4.4.1 Setup

Table 4.2 presents a descriptive statistics of our data set. We gathered 611 service description documents for SOAP-based web services and RESTful services. Each web service can contain several operations. Similarly, each RESTful service may include multiple resources. We consider an operation and a resource as a service. The services are downloaded from online sources such as WebserviceX [131], WebserviceList [130], and xMethods [141]. In total, we identify 4100 concepts from 10,100 services.

To evaluate the effectiveness of our approach to extract concepts from the service, we need to know the number of services relevant to a given concept among the 10,100 services. However, due to the limited resources, we cannot go through each of the services for a given concept. We choose six concepts with different popularity (*i.e.* high, medium and low): holiday, weather, hotel, flight, computer and camera by manually examining the service description documents. The most popular concept, holiday, is associated with 200 similar service description sets. Flight, hotel and weather are medium level of popularity and associated with 40, 60 and 40 services respectively. Camera and computer are the concepts with the lowest popularity and

associated with 10 and 8 similar services, respectively. For the sake of performance comparison, we implemented a baseline approach used by the current UDDI registry. The baseline approach groups services using a textual description of the web services and allows users to find services.

We setup a user study to evaluate the correctness of the concept recommendation and assistance during query formulation. We recruited eight participants to participate in our user study. For each participant, we gave eight different goals, such as booking a hotel room, booking a flight ticket and planning a holiday and asked them to use our prototype to find the services in order to fulfill the goals. Each participant has a software engineering background and had experience in using web services.

4.4.2 Evaluation Criteria

We use precision and recall to measure the performance of our approach to group services using concepts. Precision measures the exactness of the retrieved results and is the ratio of the total number of services correctly retrieved by our approach to the total number of services retrieved as shown in Equation 4.6. Recall evaluates the completeness of the retrieved result [76]. As shown in Equation 4.7, recall, is the ratio of the total number of services correctly retrieved to the total number of services existed. The retrieved results could be very large. It could be a tedious job for a user to go through all of them. Therefore, we use R-precision to measure the precision of top R retrieved services [76]. For example, if there are 10 services relevant to the query within the data set and 7 of them are retrieved before the 11th (*i.e.* R= 11) services, the R-precision is 70%. We choose to compute the R-precision of 10 retrieved results (*i.e.* R=10) since Silverstein et al. [108] have shown that users

mostly look at the first 10 results.

$$Precision = \frac{\{\text{relevant services}\} \cap \{\text{retrieved services}\}}{\{\text{retrieved services}\}} \tag{4.6}$$

$$Recall = \frac{\{\text{relevant services}\} \cap \{\text{retrieved services}\}}{\{\text{relevant services}\}} \tag{4.7}$$

$$R\text{-}precision = \frac{\{\#\text{ of relevant services in top } R\}}{R} \tag{4.8}$$

To evaluate the correctness of concept recommendation and assistance in query formulation in the service retrieval process, each participant determines the number of recommended concepts related to their query. Each participant also tracks if he or she reformulates their queries using the recommended concepts. We calculate the precision of the retrieved result based on the user's query. We could not calculate the recall of user's queries in the user study since it was difficult to manually check the number of services available in our data-set related to a combination of different concepts.

4.4.3 Results

Table 4.3 shows the results of precision, recall and R-precision of our approach and the baseline approach. Our approach achieves a high recall for all the services and average precision of 83%. The baseline approach has average precision of 70% and a recall of 82%. The R-precision of the top 10 retrieved results of our approach is 95% in comparison to 63% in the baseline approach. The high precision means that our approach lists relevant results in the beginning of retrieved services. Some of the services in the

Table 4.3: Comparison of the precision, recall and r-precision of our approach with the baseline approach

Concept in user's query	Our Approach			Baseline approach		
	Precision	**Recall**	**R-precision**	**Precision**	**Recall**	**R-precision**
Holiday	80%	100%	90%	67%	75%	50%
Hotel	78%	100%	90%	62%	70%	60%
Weather	93%	100%	100%	75%	95%	100%
Flight	82%	100%	90%	69%	85%	60%
Computer	85%	100%	100%	70%	83%	60%
Camera	83%	100%	100%	78%	81%	50%

hotel domain also contain "trip" in their service description documentations; hence our approach indexes the service with both concepts. The recall of our approach is 100%, meaning that our approach can effectively extract concepts from service descriptions. The low recall of the baseline approach is due to the uses of textual description in the service description documentations to group services. However, the documentation is not always available for many web services. The baseline approach uses various terms to index the same services in different concepts. For example, http://www.holidayguide.co.nz/webservices/frontdesk/holidayguide.asmx.xml has the "motel" and "holiday" in the documentation, whereas all operations are related to motel reservation.

Table 4.4 shows the result of the number of recommended concepts that the participants found useful, the number of times that participants formulate the query and the precision of the service retrieval in the user study. The user study shows that 95% of recommended concepts are related to the user's query and 85% of times a user reformulates the query using the concepts recommended. The average precision of the service retrieval is 98%. Retrieving services using a single concept give a precision of 83% as shown in Table 4.3. When a user formulates a query with a

Table 4.4: Results of concept recommendation, query formulation and precision of service retrieval from our user study

Average number of recommended concepts participants found useful	95%
Average number of times a participant reformulates the query	85%
Average precision of service retrieval	98%

specific requirement, the precision increases to 98%. Thus, we found that the query formulation helps participants to obtain a higher precision result and increases the precision of the service retrieval by 15%.

4.4.4 Threats to Validity

The main threat to our case study that could affect the generalization of the presented results relates to the number of service description documents analyzed. We have analyzed 611 services from different domains. Nevertheless, further validation of our approach requires an analysis of a larger set of service description documents. The user study is based on eight people; all of them have knowledge of web services. The validation of the concept recommendation and query formulation requires a study on a diverse group of people.

4.5 Summary

In this chapter, we present an approach of indexing services based on the semantic concepts available in the service description documents. Our approach helps to bridge the gap between the users and the service providers by recommending the concepts. We help the users formulate their web service search queries. We allow the users to combine multiple concepts, making a query more specific. We conducted a case

study and found that the average precision and recall of our approach for service discovery are respectively, 83% and 100%. Our approach categorizes retrieved results based on the concept shared between service sets. This minimizes the human effort to find a specific service. Our user study shows that the concept recommendation and query formulation make the user queries more specific and increase the precision of the service retrieval up to 15%.

Chapter 5

Quality of Experience based Service Selection

Web service composition enables seamless and dynamic integration of web services. The behavior of the participant web services determines the overall performance of a composition. Therefore, it is important to choose the high quality participants for service composition. Al-Masri et al. [4] report that there is more than 130% growth in the number of published web services. A similar observation can be made by reviewing the statistics from web service search engines such as Seekda [107]. In particular, Programmable web directory [100] indicates an exponential increase in the number of web services over the last three years. Such rapid growth in the number of services increases the importance of the service selection task due to the presence of low quality services. The state of the art in service discovery and selection relies on non-functional aspects also known as Quality of Service (QoS), *e.g.*, response time and availability. Though these parameters are crucial for selecting web services, they do not reflect the end user's perspective on quality.

In this chapter, we explore the feasibility of adopting the perceived quality from an end user's perspective for service selection and composition. We name such a set of quality parameters as Quality of Experience (QoE). We propose a solution that

automatically mines and identifies QoE attributes from the web. We also study the application of such dynamically extracted QoE attributes for service selection.

5.1 Motivation

Most research in QoS-based service selection [21, 22, 102, 146] focuses on proposing a comprehensive pre-defined QoS language to describe service requests and offers, or on implementing a selection algorithm to achieve an optimized composition. However, the process of obtaining the QoS information is largely overlooked. There are mainly two ways to obtain the QoS information: static release, and run-time monitoring. Static release of the QoS information is conducted by the service providers. The static release is not frequently updated and is done in a specific environment and platform. The posted QoS information may be different if the same service is invoked from a different geographical location or through different devices. Hence, the static information is less reliable. Run-time monitoring is the main way to collect objective and effective QoS information. Run-time monitoring approaches require analysis of web service quality at the client-side. Client-side evaluation of real world services is resource intensive, time consuming and expensive [4]. This issue threatens the applicability of the QoS-based service selection approaches [75, 102, 146].

An alternative source of information about the quality of web services is online reviews available on the web. Web 2.0 user-oriented content generation approach has enabled people to broadcast their knowledge and experience to the public. Online user reviews are one example of such a phenomenon. End users express their experience via online reviews to reveal their satisfactions and disappointments about specific services. In this chapter, we explore the possibility of exploiting user reviews

for service selection applications. We propose the concept of Quality of Experience (*i.e.*, QoE) which measures customer satisfaction with a service. QoE attributes are extracted from online reviews reflecting user experience feedback on web services. Extracting QoE attributes from the user reviews is challenging. The user reviews are written in natural language and presented as unstructured data. Therefore, it is not a trivial task for computers to understand, analyze, and aggregate QoE from the web. In our research, we present the results of our study on the possibility of automatic QoE extraction from the user reviews. We also explore the relationship between the overlap attributes between QoS and QoE. Finally, we study if QoE can replace QoS for service selection in case of insufficient QoS.

5.2 Quality Information

Service providers offer the quality metrics about the service non-functional quality. These metrics are mainly targeted for the service composers and the developers. In addition to that, after an experience with a service, a user generally provides his feedback in the form of reviews or comments. This experience with a service is called quality of experience (QoE).

5.2.1 Quality of Service

By QoS, we refer to the non-functional properties of web services such as performance, reliability, and security. Delivering QoS on the Internet is a critical and significant challenge because of its dynamic and unpredictable nature. QoS covers a whole range of techniques that match the needs of service consumers with those of the service providers based on the network resources available.

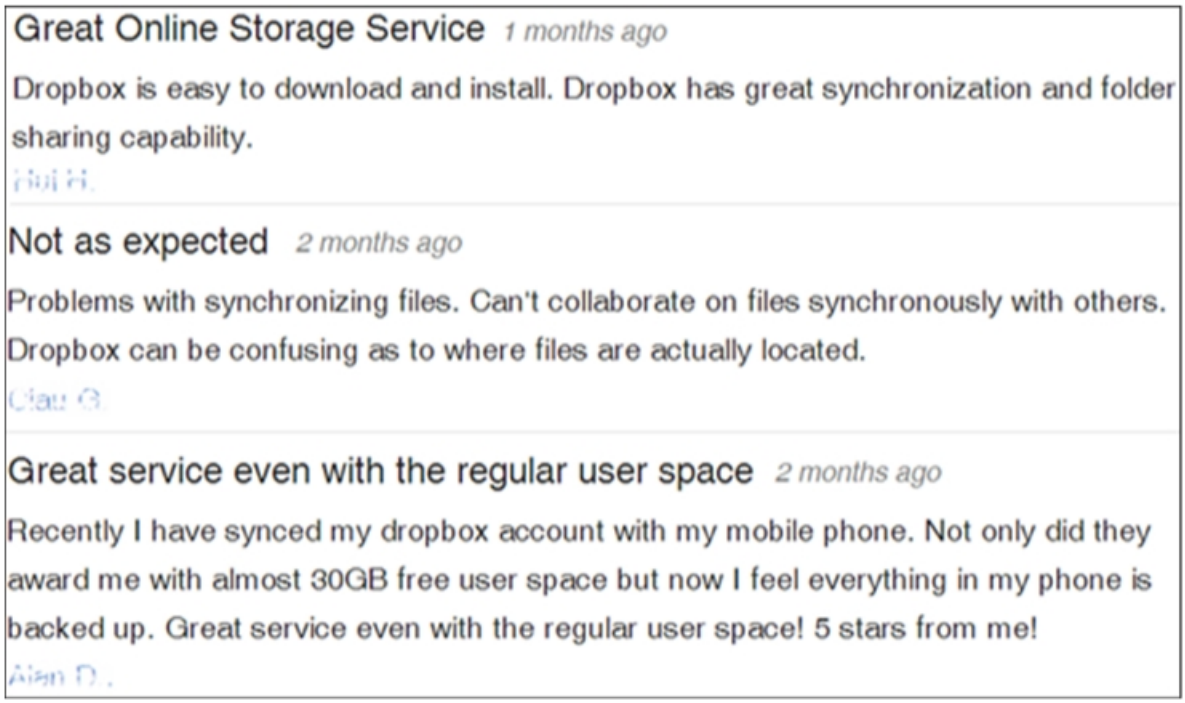


Figure 5.1: Sample reviews of an online storage provider (Dropbox)

As an ad-hoc industry standard, Service Level Agreement (SLA) is widely used to define a formal contract associated with a web service between a service consumer and a service provider, aiming at specifying quantifiable issues under specific contexts based upon mutual understandings and expectations. SLA can thus be used to define any service related issue, including QoS factors. To date, several SLA specifications and proposals are available. Among them, two popular ones are: Web Services Agreement Specification (WS-Agreement) [10] and Web Service Level Agreement (WSLA) [64]. They both define their XML-based languages and protocols for service providers to advertise web services.

5.2.2 Quality of Experience

Quality of Experience (QoE) is a subjective measure of an end user's experiences with a service. A service has different aspects, *e.g.*, cost and performance, for which an end user providers his opinion. Each aspect of a service is a QoE attribute. Contrary to QoS, QoE is reflecting quality from the end user's point of view. The primary source of QoE is online reviews. Reviews come from users with diverse platforms and from different geographical locations. Hence, it is a credible source of information. Figure 5.1 shows a review on a service by three different users from a website (http://pcmag.com). The reviews contain valuable information provided by people who used the service. The first user is telling about his experience with "synchronization" and "folder sharing" capability. Similarly the second user is expressing her dissatisfaction with "synchronization", "cross platform support" and "security". These attributes can be directly mapped to QoS parameters such as "performance" and "security".

Users use natural language to provide their feedback. In feedback, a user may mention more than one quality attribute of a service. Consider a user is looking for an online storage provider. He wants to find the service that have the best cost and works great for media streaming purpose. QoE attributes of web services are not readily available and distributed in different social media and review sites. Obliviously without the automatic aggregation and search tools, finding and going through a large number of review to select a service is time consuming and tedious. Moreover, these informational cannot be directly used by service composition engines to recommend and select a service.

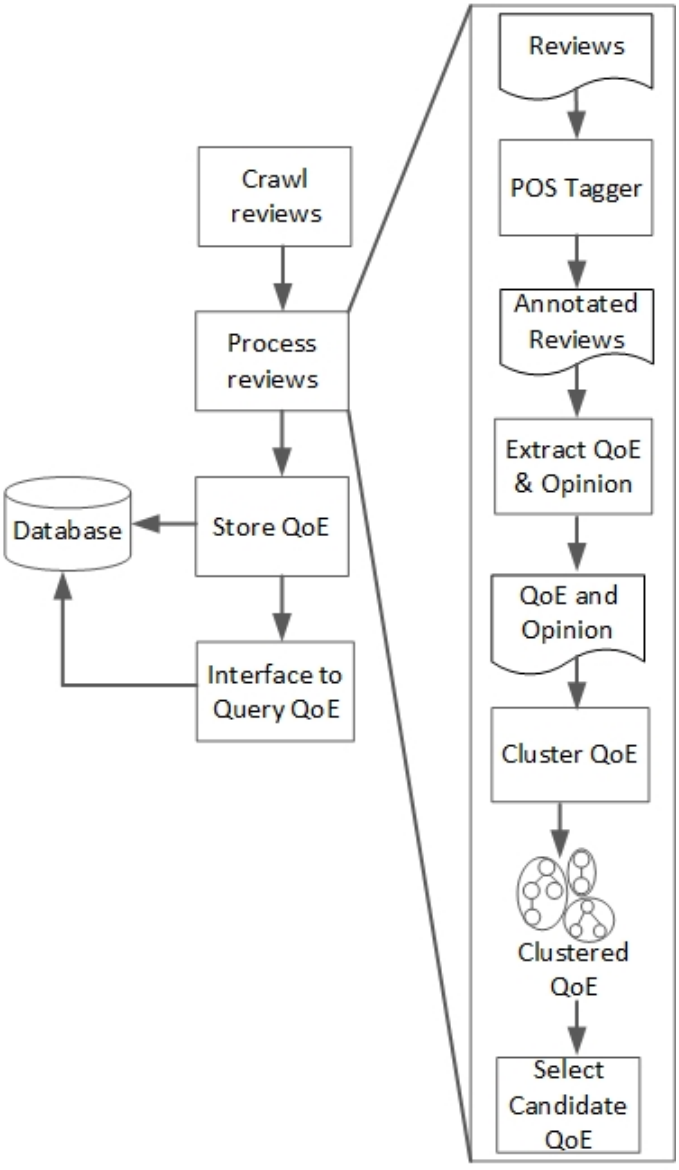


Figure 5.2: Approach to extract QoE attributes

5.3 Our Approach to Extract QoE Attributes

In this section, we describe our approach to extract quality of experience information for web services. Figure 5.2 shows an overview of our approach. Our QoE extraction approach mainly consists of four steps. First, we crawl the web for user reviews. In the

next steps, we use natural language processing techniques to dynamically extract QoE attributes. Finally, we provide an interface to query the extracted QoE attributes for service selection.

5.3.1 Crawling Online Reviews

Given an unseen web service, we crawl reviews and put them in the review database. We form a web search query to get the reviews posted within the last 2 years on the Internet. The downloaded reviews are locally stored as HTML web pages. Malformed HTML files are quite common in the web. For example, an HTML file may contain mismatched HTML tags. To generate the DOM tree structure from an HTML file, we use the HTML syntax checker [60] to correct the malformed HTML tags. We then extract reviews from the stored pages in a text format without HTML tags.

5.3.2 Processing Reviews

A review typically comprises of several sentences. Usually, a single review by a user expresses multiple positive and negative opinions. For example, a Dropbox reviewer may use a couple of sentences to praise its performance, but use other sentences to belittle its cost and media streaming capability. A QoE has two major data fields which are attributes and opinion associated with an attribute. For each review, we identify the target attribute and opinion. It is not trivial to determine the opinion orientation of such a review as a whole. To overcome this problem, we split a review into sentences. This approach makes it possible to assign positive or negative opinions on different aspects of an experience.

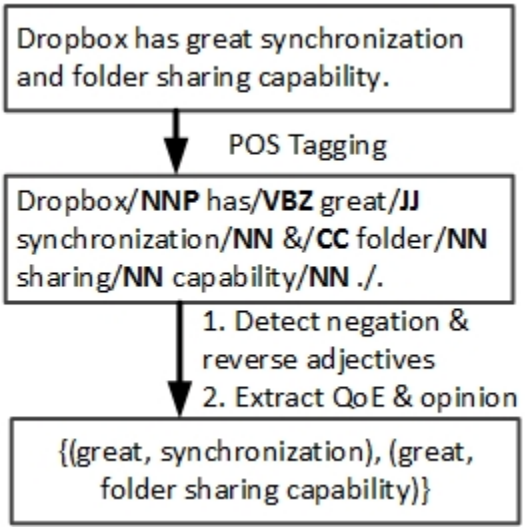


Figure 5.3: Extracted QoE attributes and opinion based on POS

Tagging POS in Reviews

Natural language processing helps us determine the part of speech (POS) of each word in a sentence. POS is used to define a syntactic or morphological behavior of a word. The English language grammar classifies parts-of-speech in the following categories: verb, noun, adjective, adverb, pronoun, preposition, conjunction and interjection. Each of the above mentioned categories plays a specific role within a sentence such as nouns give names to objects, and an adjective qualifies a noun. As a result, POS identifies the behavior of each word which in turn helps us understand a reviewer's experience. We use a well-known POS (part-of-speech) tagger [120] to identify the syntactic structure of a sentence. The second box in Figure 5.3 shows review sentences with POS tags. We post-process the generated tags to resolve object names consisting of multiple words (*e.g.*, "Folder sharing capability"), phrasal verbs (*e.g.*, "go to"), and pronominal referrals (pronouns *e.g.*, "it"). We assume cases like "it" always refer to the last mentioned object, which proved to be a sensible heuristic in most of the cases.

$$Review = (service, user, date, body, (QA, RQA), TV) \tag{5.1}$$

where, body is the text content of a review from a user on a specific date for a service. QA and RQA is the quality attribute and its rank provided by the user. TV is the overall value for a service

Extracting QoE Attributes and Opinion

We model reviews as shown in Equation 5.1 and transform extracted review to a model shown in Equation 5.2. For a review quality attributes (*i.e.*, QA) and its rank (*i.e.*, RQA) are stored as QoE and OScore in Equation 5.2. In addition, we extract quality attributes from the body of a review by analysing its POS.

$$Extract_{Review} = \{QoE, Opinion, OScore, Date\} \tag{5.2}$$

where QoE is a quality of experience attribute; Opinion is the opinion about QoE; OScore is the polarity score of Opinion and date is the time when the review was posted.

Nouns give names to objects, beings or entities in the domain of discourse. Adjectives and adverbs reflect the opinion about nouns. Opinions encode an emotional state, which can be desirable or undesirable. Opinions that encode desirable states ("beautiful", "nice", and "happy") have positive orientation while the ones that encode undesirable state ("bad", "terrible", and "disappointing") have a negative orientation. Opinions are usually adjectives and adverbs. Often the opinion information in a sentence is expressed as "not", "no", and "barely". In such case, the sentiment about the QoE attribute is the opposite of the corresponding opinion phrases. For

example, two consecutive negative terms reflect a positive opinion (*e.g.*, no problem). The overall idea is to apply such rules to infer the final value for each mentioned QoE attribute. We employ the idea proposed by Turney et al. [122], where two consecutive words are extracted from the review if their tags conform to predefined patterns. The first pattern means that two consecutive words are extracted if the first word is an adjective and the second is a noun. For example, "The map supports multiple destinations", the "multiple destinations" phrase is the quality. The second pattern means that two consecutive words are extracted if the first word is an adverb, and the second word is an adjective, but the third word is a noun. The third pattern means that two consecutive words are extracted if they are all adjectives, but the following word is not a noun. Singular and plural proper nouns are avoided so that the names of the items in the review cannot influence the classification. At this stage, we extract QoE attributes and opinion of each review. We store the extracted information in a tuple shown as shown Equation 5.2.

We quantify the QoE attribute based on the opinion provided by end users. In this research, QoE can be qualitatively scored between [0,1] using real numbers. 1 represents the highest positive opinion for a service, and 0 relates to the lowest negative feedback. We used SentiwordNet [41] to calculate the positive and negative effect of an opinion in the QoE attribute.

Clustering QoE Attributes

At this stage our responsibility is to find related attributes which are presented with different phrases and find a candidate title for each group of similar candidates. An extensive list of QoE attributes and opinions of QoE attributes extracted using the

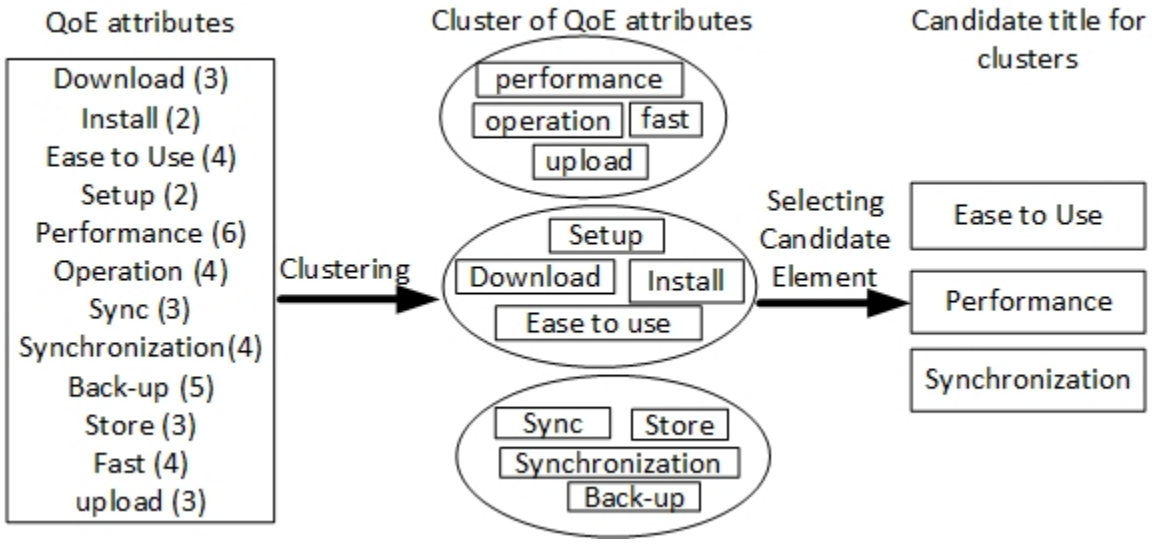


Figure 5.4: Process of clustering QoE attributes and selecting a candidate element

process defined in Section 5.3. QoE attributes are not predefined since they depend on the nature of target web services and end users experience. Our aim in this step is to group similar QoE attributes together and summarize the opinion on the finalized QoE attributes. To automatically create the clusters, we use k-means [44, 54] which is an unsupervised clustering algorithm. Clustering algorithm, in our case k-means, divides the data into a set of disjoint groups.

The main challenge in using such clustering algorithm is to identify the expected number of clusters [104]. In case of k-means, this parameter is called k. One possible solution is to ask domain experts to identify the proper value for k empirically. However, since we need to automate the process completely, we use a clustering validation approach proposed by Rousseeuw [104]. Using this approach, we can measure the success of any possible value for k in generating a set of coherent clusters. To find the proper value for k automatically, we create clusters with all possible values for k where the maximum value is the number of distinct data points. Then, we measure

the success of each experiment using Rousseeuw [104] approach. Finally, we select the k value with the highest measured success rate for our actual clustering step in Figure 5.4.

$$wordSim(x, y) = 1 - \frac{mcp(cp)}{mcp(cp) + dcp(cp, root)} \tag{5.3}$$

where cp is the common parent of the two QoE attributes x, y; root is the root of the WordNet ontology; minimum common parent length (i.e., mcp(cp)) is the shortest path from either x or y to cp, and dcp(cp, root) is the length of the path from cp to root.

We use semantic similarity as shown in Equation 5.3 to find the distance between words. We use WordNet [86] to find the similarity between the QoE attributes. In WordNet, all words are connected as a graph. The two words can be directly or indirectly connected through many intermediate relations. The distance in our approach is defined as the number of intermediate words of the shortest path between two words. The similarity between two words x and y is measured by the path length (*i.e.*, dcp in Equation 5.3) between words to reach their common parent in the WordNet ontology as used in [86]. The value of the similarity shown in Equation 5.3 ranges from 0 to 1. 0 represents unrelated words and 1 signifies synonymous words. Figure 5.4 shows the extracted QoE and the corresponding clusters based on the word similarity. In total, our approach identified 3 final QoE attributes (shown as clusters) from initial 8 QoE attributes.

In this step, we identify a representative QoE attribute for each cluster of QoE attributes. The candidate element represents the whole cluster and its sentiment associated with the cluster. Our approach to select a candidate element from a

cluster is similar to the approach described in Chapter 4. Equation 5.4 shows how we compute the rank of a QoE attribute x in cluster C.

$$R(x) = \{ \sum_{y \in C; y \neq x} WordSim(x, y) f(y) \} + f(x) \tag{5.4}$$

where R(x) denotes the rank of the QoE x in the cluster C; WordSim(x, y) is the similarity between QoE attributes x and y, and f(x) is the frequency of the QoE attribute x.

Ranking QoE attributes signifies the importance of a QoE attribute with respect to the other QoE attributes in a cluster. The computed rank is then normalized between 0 to 1 by dividing the raw value by the sum of all QoE attributes rank values in a cluster. 1 signifies the most dominant QoE and a QoE with the largest normalized rank value represents the cluster. For example, in Figure 5.4, the similarity between "sync" and "synchronization" is 1 as one is the abbreviation of another; "synchronization" and "backup" is 0.7; "synchronization" and "store" is 0.6. Using these similarity values, we compute the rank of the QoE attributes {synchronization, backup, store}, as {synchronization (0.3+0.6+7=7.9), backup (0.3+0.4+5=5.7), store (0.6+0.2+3=3.8)}. Hence, we select a representative QoE attribute Synchronization for the cluster {Sync, Store, Synchronization, back-up}.

5.3.3 Storing and Querying QoE Attributes

Once we have ranked and indexed services based on the user's quality of experience. We store the QoE attributes in a database. We provide a RESTful interface on top of the database. A user has the ability to query for QoE attributes for a service. The result shows information about services such as the name of a service, service category

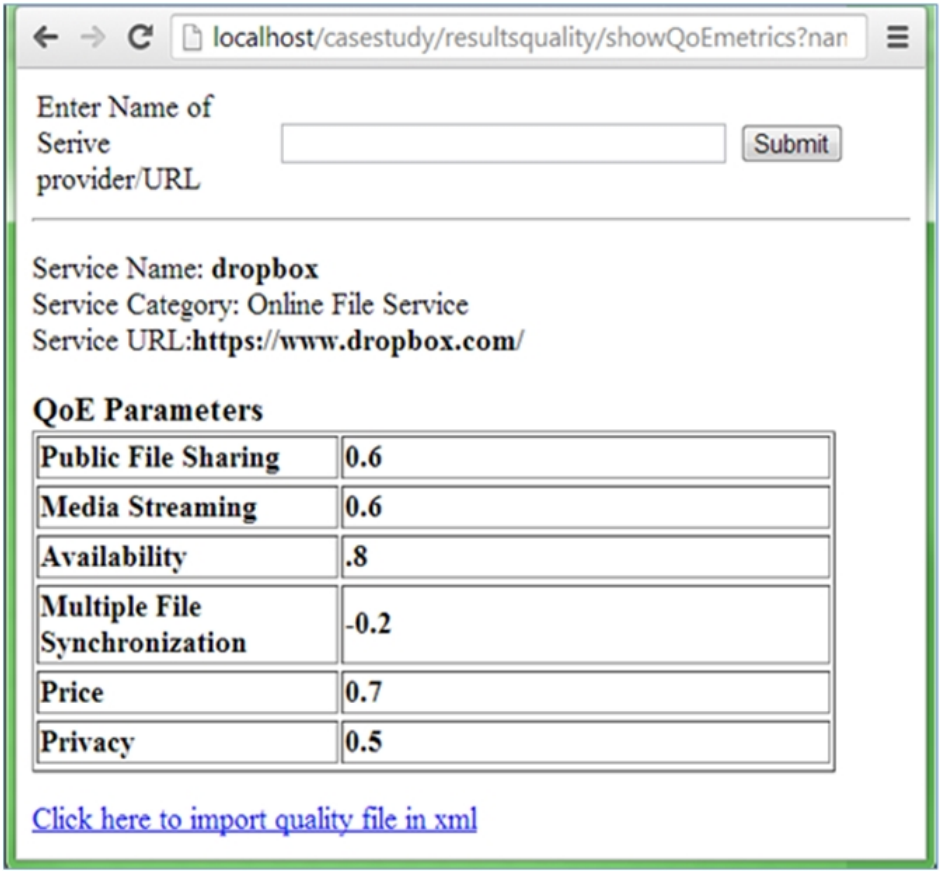


Figure 5.5: Interface showing QoE related to a service

and QoE attributes and its score as shown in Figure 5.5. A user can also query about the trend for each QoE attribute. QoE attributes and opinions are recalculated and updated as new reviews are downloaded by the crawler.

5.3.4 Selecting Services and Service Execution Paths

For a composition, different structures (such as sequential, loop) can be used to connect the services. We focus on compositions that are defined using sequential and conditional structures. Other structures, such as loop, for example, are reduced to sequential. Our focus is to identify services based on user preferences. A user has

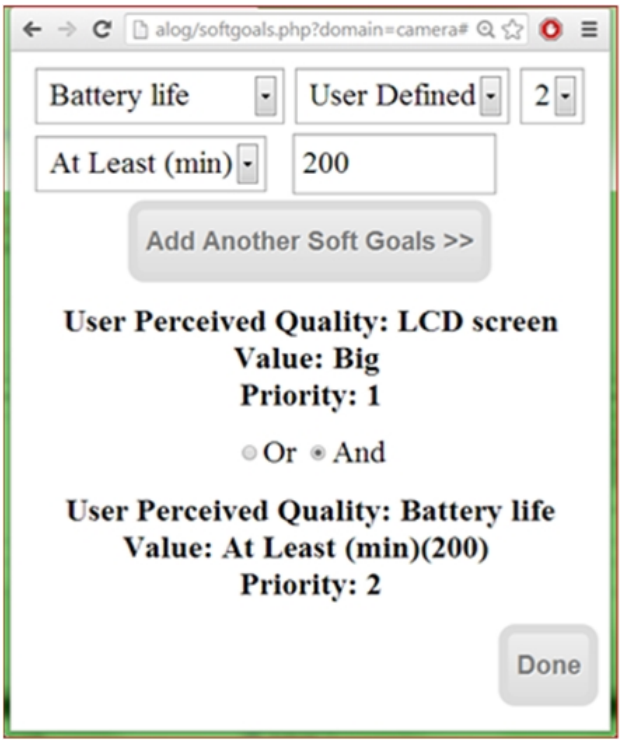


Figure 5.6: Interface showing a user selecting different quality metrics

multiple preferences as shown in Figure 5.6. A user provides multiple preferences, which contain {QoE attribute, value, priority}. Multiple QoE attributes are connected through logical connectors such as "AND" and "OR". An "AND" connector between two QoE attributes means both the services should have to fulfill the criteria. An "OR" connector between two QoE attributes means one of the services should fulfill the QoE attribute. A priority helps a user to prioritize a QoE attribute among two QoE attributes connected through "OR" connector. Moreover, a composite service can be represented by a statechart which has multiple execution paths when containing conditional branching. For each task, there can be multiple services that can fulfill the objectives. At the initial stage, the selection of a service from a pool of services is done based on the QoE attribute of each service. In Figure 5.7, the

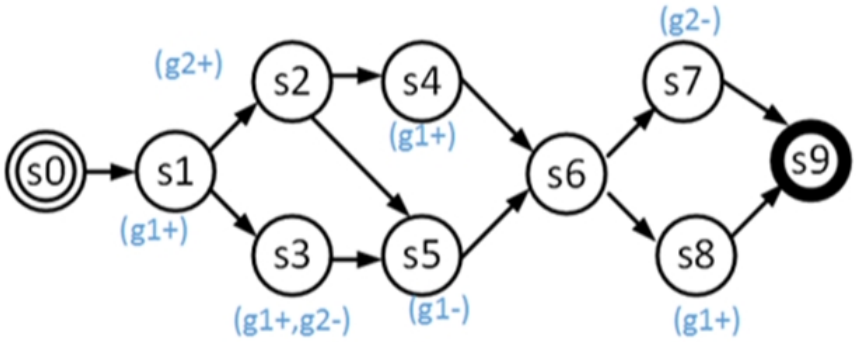


Figure 5.7: Web service composition as a state chart

services such as “s1”, “s2”, and “s3” is based on the values of QoE attributes “q1”, and “q2”.

Figure 5.7 shows the web service composition example. Each node represents a service. Nodes “s1”, ”s2” and “s6” have the branches. Each node has a QoE value written such as “g2+”, “g1-”. “g1” and “g2” are the quality metric associated with the services. The +/- sign indicates the polarity of a service for a quality metric. Each execution path represents a sequence of tasks to complete a composite service execution. Furthermore, for a composite web service, we notice that we can have different possible combinations. Our work aims at advancing the state of the art in web service composition by using the user’s preferences for selecting the services and the execution paths.

Figure 5.8 shows different execution paths derived from the service composition in Figure 5.7. The services in Figure 5.8 have two QoE attributes (*i.e.*, “g1” and “g2”), “gtotal1” and “gtotal2” are the sum of the QoE attributes for each path. Based on user’s preference, services are selected. A user’s preference during service selection is also used during the execution path selection. Figure 5.8 shows an interface to select QoE to buy a digital camera.

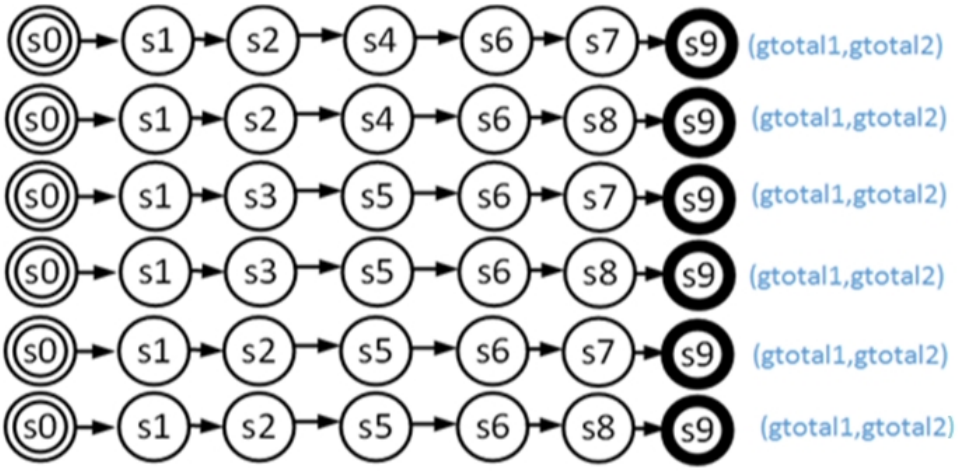


Figure 5.8: Showing the combination of different execution paths

Figure 5.8 shows all the execution paths derived from the composite service as described in algorithm shown in Figure 5.9. We use breadth first algorithm to find all the execution paths for a composite service as shown in the algorithm in Figure 5.9 (*i.e.*, "getAllPaths()" in line 1). In order to find the optimal path in a DAG (Direct Acyclic Graph) structured composite service, we compute the overall QoE for each path as shown by "computeOverallQoE" (line 4) for each path p and stores in a hashmap (*i.e.*, "QoEpaths" in line 5) as shown in Figure 5.9. Finally, the algorithm compares and returns the optimal execution path.

5.4 Prototype of Our Service Recommendation Approach

In this section, we describe the prototype of our tool. Figure 5.5 displays an interface to query QoE attributes for a service. A user can search by the name of services or provide the URL of a service. Lower part of Figure 5.5 shows the result of a query. Figure 5.6 illustrates an interface for a user to select a list of QoE attributes to include in service composition. Figure 5.10 demonstrates the process to purchase

```
Input: G //DAG Graph of Composite Service
       UserPerceivedQoE //User provide QoEs and connector between QoEs
Output: Execution path
Process:
1.    Paths[]=getAllPatths(G); //Get all the paths from Graph G
2.    HashMap<Path,QoE> QoEpaths;
3.    For each path p in Paths
4.          QoE=computeOverAllQoE(p)  //compute the total QoE for each path
5.          QoEpaths.put(p,QoE)
6.    End For
7.    Path optimalPath=QoEpaths.get(0)
8.    For each path p in QoEpaths
9.          If(CompareUtility(optimalPath, UserPerceivedQoE, p)
10.               optimalPath=p
11.         End If
12.   End For
13.   Return optimalPath
```

Figure 5.9: Algorithm to identify an optimal path

a product. A user has two choices as shown in Figure 5.11 1) select a used product or a new product; or 2) select an express delivery or a regular delivery. Based on user preferences, our approach selects the best available services and recommends the optimal execution path to meet the desired QoE attributes. Figure 5.11 shows two scenarios (*i.e.*, Scenarios #1 and #2). In both scenarios, a user chooses *iPhone 3S* as a product.

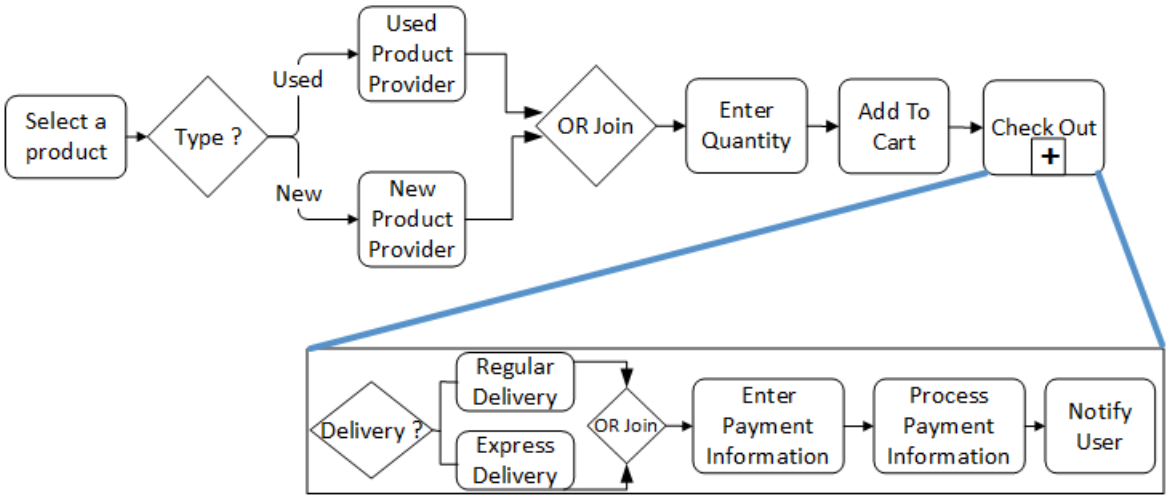


Figure 5.10: A business process to buy a product

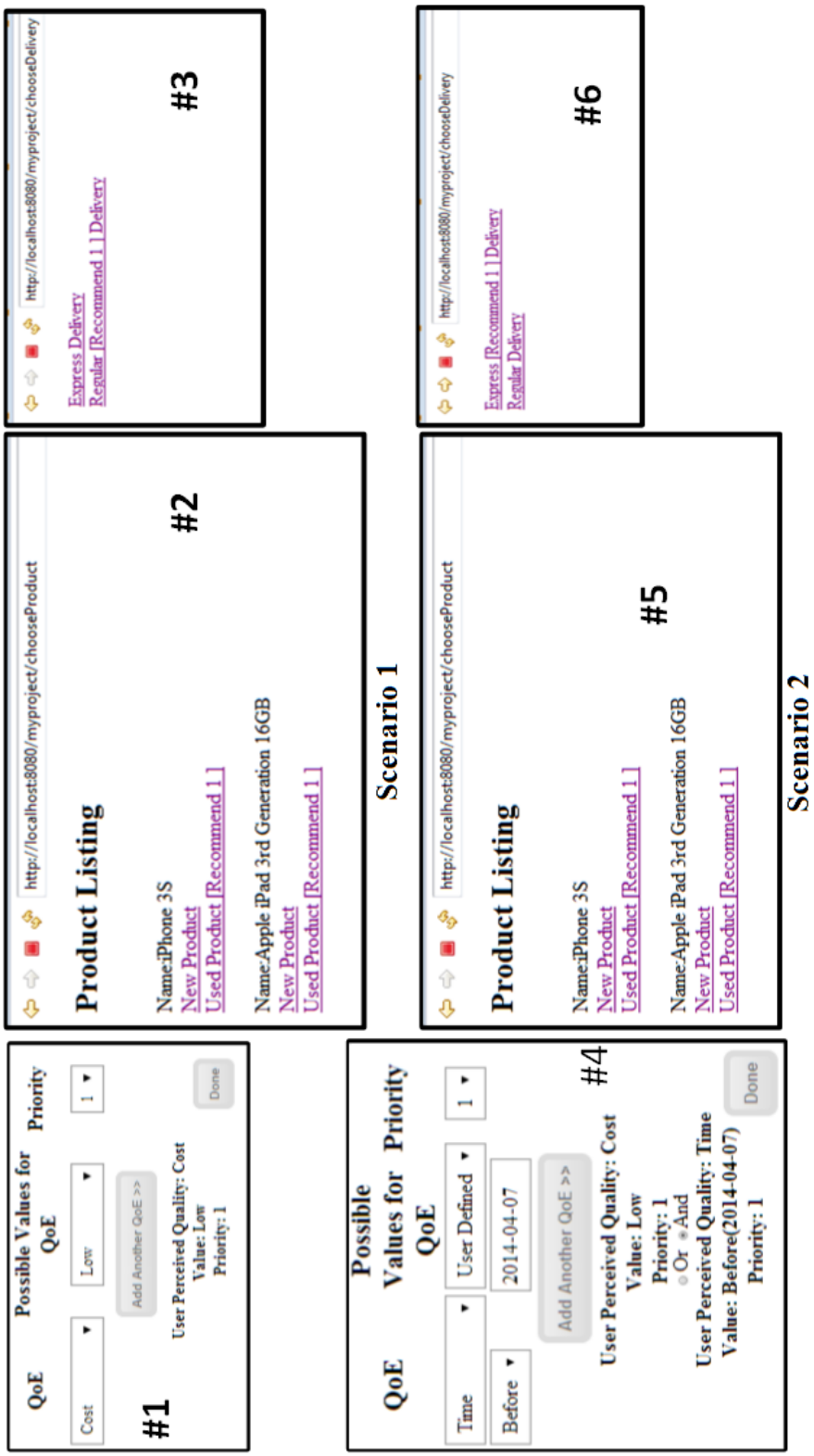


Figure 5.11: Scenarios showing service recommendation by our approach

In Scenario #1, a user wants to buy an *IPhone 3S* at a low cost. Hence, a user selects a QoE attribute "Cost" and assigns "low" as its value. Our approach first selects services that provides lower cost *IPhone 3S* and recommends "Used Product" as shown in #2 of Figure 5.11. Similarly, we find services that can deliver the product at a low cost. Between "Regular Delivery" and "Express Delivery", our approach chooses "Regular Delivery" as shown in #3 of Figure 5.11. For Scenario #2, a user selects two QoE attributes "Cost" and "Time". A user enters "low" as the value for "Cost" and "2014-04-07" as the value for "Time". The scenario is run on the "4th of April 2014", and the user location was set to "Kingston, Ontario". Based on the given QoE attributes, our approach suggests buying the "Used Product" and use "Express Delivery". Even though QoE attribute "Cost" is "low", "Express Delivery" is recommended to meet the time constraints.

5.5 Case Studies

We conduct a case study to evaluate the effectiveness of our approach. The objectives of the case study include: 1) evaluating if our approach in terms of precision and recall for automatic QoE extraction; and 2) examining the correlation between the Quality of Service and the Quality of Experience.

5.5.1 Data Collection and Processing

We collect reviews for web services from four different domains, 1) trip (*e.g.*, CleanTrip and Ebookers), 2) shopping (*e.g.*, Amazon and eBay), 3) storage (*e.g.*, Dropbox) and 4) mapping service (*e.g.*, Google Maps) as shown in Table 5.1. The services in the first two domains are aggregators, which are brokers that package and integrate multiple

Table 5.1: Services and their review sentences used in our case study

Domain	Agg.	# Services	Sentence in reviews	Sentences with QoE & opinion
Trip	Yes	Yahoo Travel, Expedia, Tripit, Hotwire, Belair, Cleantrip, Ebookers	7428	6980
Shopping	Yes	Amazon , eBay, Best Buy, Zappos , Checkout, Discfoo	6306	5866
Storage	No	Dropbox, Sugar Sync, Google Drive, Sky Drive Box	7033	6611
Mapping Service	No	Yahoo Maps, Google Maps, Bing Maps, Open Street Maps	4529	4110

web services into one or more composite services. To avoid skewness in the data, we crawled similar number of reviews for each category. The reviews were crawled from pcmag.com, sitejabber.com, Cnet.com, programmable web and expert reviews. For each service, we crawled and downloaded reviews. We clean these reviews by removing HTML tags and store the review in the format as discussed in Equation 5.1. Table 5.1 shows the services that are considered for our case study. The table also describes the number of sentences extracted from the reviews and the number of sentences directly expressing the quality of experience. We used the gathered raw data as the input of our case study.

We also manually created a gold data-set for the raw data in order to be able to evaluate the performance of our proposed approach in Section 5.3. Therefore, in our case study, we inspected all the data to create the gold data-set for QoE attributes. To create such oracle, we manually read all the reviews. For each sentence in a review, we tag QoE related attributes and opinions. Whether the opinion is positive

or negative (*i.e.*, the orientation) is also identified. If the user gives no opinion in a sentence, the sentence is not tagged as we are only interested in sentences expressing an opinion in this work.

As part of our study, we require QoS information of the subject services. During the preparation phase, we gathered the required QoS data. We implemented the service invoker using JDK 7.0, Eclipse 3.6, Axis2 and HTTPClient4.3. Axis2 is employed to generate the web service invocation and test cases for SOAP-based services. HTTPClient4.3 is used to invoke the RESTful services. We used an automated agent to measure the average response time by considering a period of two months. We extracted the availability of services from the data posted by the service providers. We extracted the service cost and usage limits from service providers' documentation. The information regarding price and usage limits were not readily available, and we gathered them manually.

5.5.2 Evaluation of Our Approach to Extract QoE Attributes

QoS attributes are predefined, and documented (*e.g.*, [102, 146]). QoE attributes are dynamic and domain dependent. To overcome this issue, we proposed an approach to extract QoE attributes automatically from the web. We measure effectiveness of our approach to extract QoE attributes from reviews. We perform an evaluation of QoE attributes over a period of time.

Approach

We measure the effectiveness of our approach to extract quality of experience (QoE) attributes using precision and recall. We compare our approach with extracted quality

of experience attributes with the gold standard. As shown in Equation 5.5, the precision is the ratio of the total number of QoE attributes correctly extracted by our approach to the total number of QoE attributes. Recall is the ratio of the total number of QoE correctly extracted by our approach to the total number of QoE existed in the reviews as shown in Equation 5.5. However, to successfully calculate the precision and recall we require an oracle covering the relevant QoE attributes that are required by Equation 5.5 and 5.6. We use the QoE oracle for services in Table 5.2 which is created manually as part of our case study set-up.

Results

Table 5.2 summarizes the result to evaluate the effectiveness of our proposed approach in extracting QoEs automatically. We compared the extracted QoE attributes with the manually made oracle that covers all QoE from four selected domains. The effectiveness is measured via precision and recall as described in Equation 5.5 and Equation 5.6.

$$Precision = \frac{\{\text{relevant attributes}\} \cap \{\text{retrieved attributes}\}}{\{\text{retrieved attributes}\}} \tag{5.5}$$

$$Recall = \frac{\{\text{relevant attributes}\} \cap \{\text{retrieved attributes}\}}{\{\text{relevant attributes}\}} \tag{5.6}$$

Our approach extracted all QoS related QoE attributes with 100% precision and recall. The additional new domain specific QoE attributes extracted by our approach have the precision above 90% meaning that our approach can correctly identify the QoE attributes. The recall is above 79% meaning that our approach better coverage.

Table 5.2: Result of our evaluation to extract QoE attributes from online reviews

Domain	Overlapped QoE and QoS Attributes		New QoE Attributes		Total # QoE Attributes	# Overlapped	# New QoE Extracted
	Precision	Recall	Precision	Recall			
Travel	100%	100%	93%	72%	18	5	8
Shopping	100%	100%	92%	87%	16	5	9
Storage	100%	100%	93%	76%	17	5	8
Mapping Service	100%	100%	90%	82%	17	4	10

Our manual investigation revealed that the missing cases, that affect our recall negatively, are due to implicit expressions. In such cases, QoE attributes may not appear in sentences explicitly. We call such a QoE attribute as implicit QoE. For example, in one of the reviews related to mapping service, the reviewer expressed her unsatisfactory opinion about the latency time by saying "you can go for a cup of tea after requesting ...". In overall, considering the limitations in the opinion mining domain and comparing to the performance observed in the other successful opinion mining of other domains (*e.g.*, [122]), we can conclude the performance is of our approach is acceptable.

As shown in table 5.2, our approach identified more than twice as many quality attributes (*i.e.*, total number of QoE attributes) as those that present in traditional QoS attributes (*i.e.*, total number overlapped attributes). We selected eight most frequent extracted attributes and plotted the values for the past 13 months in Figure 5.12. Figure 5.12 shows each service has their own weak and strong aspects. Allowing a user to select a service based on different aspects gives him more satisfaction. Figure 5.12 also shows another important aspect, the change of quality attribute value over time. We see three different kinds of trend. The first trend is the user's sentiment

of a particular QoE attribute remains almost constant over a period of time as in the case of "Ease of Use" and "Cost" QoE attributes. The second trend is a QoE attribute of a service is high during the initial phase and it slowly demises and again gains the user's confidence as seen in case of Dropbox and Google Drive for "Media streaming" QoE attribute. The third trend is a slow and steady rise or fall in the user's experience. The rise of the third kind of trend is seen in "Mobile Access" QoE attribute of skydrive and sugarsync services. The fall is seen in file sharing attribute for Box service. The fourth trend is the oscillating trend, which is high for certain duration and low in certain duration as seen in the case of Google drive in Figure 5.12. Our result shows the QoE attributes are enhanced and capture dynamic and domain dependent aspects of different services.

5.5.3 Evaluation of Correlation between QoE Attributes and QoS Attributes

QoE and QoS come from different sources. The process of collecting QoS related information is tedious, time consuming and difficult to collect at the client-side [4]. QoS is provided by the service providers or recorded by the client, whereas QoE is directly based on the user's feedback. In this study, we explore the option of using QoE during the service selection process. Since our approach can be automated, and it is independent of a service provider. A strong correlation between QoE and QoS attributes indicates the possibility of using for service selection.

Approach

To evaluate the relation between the QoS and QoE attributes, we collected QoS attributes for all the services based on [4, 102]. We measured and collected each of the quality metrics described in Section 5.3 for which we have QoE attributes. We manually mapped different QoE attributes to corresponding QoS attributes. For example, QoE attributes "synchronization", "upload Speed", and "media streaming" in the storage domain are mapped to QoS attribute "performance". QoS attribute "performance" for storage was measured as the time to upload a file or retrieve the uploaded file. To study if opinion expressed by QoE attribute is in agreement with QoS data, we use the Pearson correlation coefficient. The Pearson population correlation coefficient of QoS and QoE is defined as the ratio of the covariance of QoS and QoE and the product of their standard deviation.

Results

Our approach discovered five QoS attributes (Performance, Availability, Usage Limit, Security and Cost) in the reviews. Security is excluded because QoE attribute "Security" corresponds to the number of times a user felt the system or software were hacked or broken. But this information was not freely available. As security, the QoS information available were the encryption and secure socket layer used by the service provider. Since we cannot measure it, we decided not to use the metric in our study. Similarly, we did not find QoS attribute cost for travel, shopping and mapping service, as all of the services in these domains are free. Table 5.3 lists the correlation, fitness and p-value of related QoS attributes and QoE attributes. Our study shows a high correlation between QoS attributes and QoE attributes except in

the case of QoS (cost) in storage domain. Performance is negatively correlated as less response time (QoS) is better while higher value (closer to 1) of QoE performance means people are saying well about performance. We also found difficult to relate different aspect of the same parameter. For example, QoS attribute cost for a service provider is about the service invocation cost. Whereas, the QoE attribute cost in case of an end user means the cost of a service or a product. We found the availability of a shopping service and availability of a trip service do not have the same level of correlation as other QoS attributes. We went and re-analyzed the sentiment related to availability for shopping service. We found the sentiment of availability was mixed with product availability and service availability. Similarly, for trip services sentiment for availability is mixed with both the hotel and flight availability.

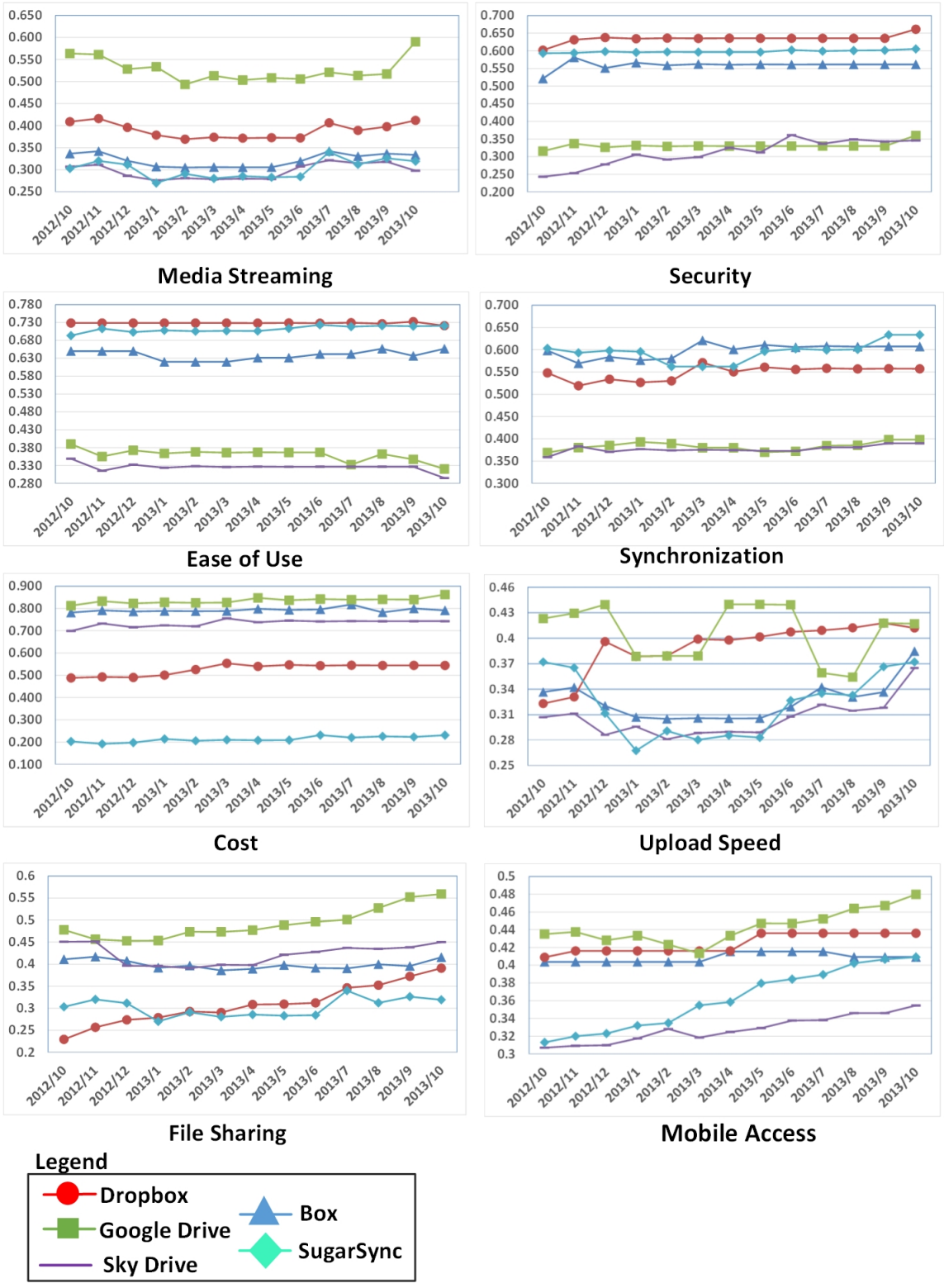


Figure 5.12: Eight most frequent QoE attributes of online storage providers over a period of 13 months

Table 5.3: Relation between QoE attributes and QoS attributes

Domain	**Performance**			**Availability**			**Usage Limit**			**Cost**		
	cor.	**r^2**	**p-value**	**cor.**	**r^2**	**p-value**	**cor.**	**r^2**	**p-value**	**cor.**	**r^2**	**p-value**
Travel	-0.948	0.900	**0.001**	0.475	0.226	0.280	0.994	0.989	**3.6e-6**	-	-	-
Shopping	-0.939	0.883	**0.005**	0.333	0.111	0.518	0.986	0.973	**0.01**	-	-	-
Storage	-0.950	0.904	**0.012**	0.968	0.937	**0.006**	0.998	0.996	**3.7e-6**	0.01	0.002	0.97
Mapping Service	-0.953	0.908	**0.046**	0.994	0.988	**0.005**	0.713	0.508	0.286	-	-	-

The cost related QoS in table 5.3 is online storage and there was almost no correlation between the QoS attribute "Cost" collected and sentiment of QoE attribute "Cost". When we analyzed the reviews related to cost for online storage, we found most of sentiment were related to the free storage space rather than the commercial plan of storage. However, we find the correlation between free space provided by the service provider and the sentiment of QoE attribute "Cost". Our analysis shows the correlation between QoE attributes "Cost" and "Free space" is 0.946 and the fitness value is 0.895. Hence, reviews and comments on online storage were based on free storage rather than average storage.

5.5.4 Threats to Validity

In this sub-section, we discuss the limitations of our approach and the different types of threats which may affect the validity of the results of our case study. The main threat to our case study that could affect the generalization of the presented results relates to the number of service description documents analyzed. We have analyzed more than 24,000 reviews of different services from different domains. Nevertheless, further validation of our approach requires an analysis of a larger set of reviews. Our data-set was limited to reviews from last 2 years for a list of review websites and hence does not give the whole picture of all the comments by a user. The QoE attributes is manually checked, and it is arguable whether a particular attribute is a QoE attribute or not.

5.6 Summary

In this chapter, we present a framework to identify and aggregate QoE attributes of a service. Our case study shows significant performance on the identification and grouping of QoE attributes in reviews. We provide an approach to query the quality attributes for a service. Since all the steps are performed in a domain-independent way, our approach is flexible enough to be equally applicable to any other domain. Though, the recall of QoE identification is not high, in a real life scenario, most of the services have a sizeable number of reviews, and hence even a moderate recall result could be representative and helpful to the customers. We found both the QoE and the QoS attributes are highly correlated, suggesting that we can use QoE attributes for service selection. We provide an approach for a user to choose QoE attributes. Our approach selects the services and the execution paths based on the user selected QoE attributes.

Chapter 6

Extraction of Process Knowledge and Service Linkage

Software is prevalent in all aspects of our lives, such as checking a stock price, finding a doctor and buying a product. Nowadays a significant part of any software system is structured using software services and implemented using web service technologies. Current approaches in Service-Oriented Architecture (SOA) are challenging for the users to get involved in the service composition due to the in-depth knowledge required for SOA standards and techniques. Process knowledge, such as tasks involved in a process, the control flow and the data flow among the tasks, is essential for designing business processes. Such process knowledge enables service composition which integrates different services. In the current state of practice, business processes are primarily designed by the business analysts who have extensive process knowledge. It is challenging for the novice business analysts and the end users to identify a complete set of services to orchestrate a well-defined business process due to the lack of process knowledge.

In this chapter, we propose an approach to extract process knowledge from the

web. Our approach uses a web search engine to find the websites containing the process knowledge. To shield the users from the complexity of SOA standards, we automatically generate composed services for end users using process knowledge available in the web. Our approach uses the natural language processing techniques to extract the tasks. We represent the extracted tasks in a task model. We use the task models to find the services and then generate a user interface (UI).

6.1 Motivation

In the current state of practices, service composition is based on a predefined process model that describes the services needed to accomplish a task, such as planning a trip. A significant amount of effort from industry and academics focuses on providing infrastructures, languages and tools to compose the services. However, the complexity of web services technology prevents users with limited IT skills from getting easy access to web services and their offered functionalities. A user has to perform a sequence of tasks to complete a process. From the implementation point of view, a task can be implemented by one or more services. For example, a typical process of buying a movie ticket includes tasks related to "searching for movies", "choosing the date and time", and "paying for the ticket". Web service composition [47, 51] requires the resolution of multiple dependencies between the input parameters (IP), the output parameters (OP) and non-trivial reasoning about the composition of required functionalities from smaller units of web services. It is challenging to acquire the complete knowledge of a domain (*e.g.*, hotel booking, flight booking in travel) and then to search, and combine the services found. We envision two challenges for a novice designer or a user to perform service composition (SC) as listed below:

Lack of complete knowledge about the tasks involved in order to accomplish a goal. A user repeatedly searches the web to learn and complete different tasks required to achieve a goal. For example, searching for a movie, finding the showtime in a local theater and making online payment are tasks for buying a movie ticket. This process is tedious and time consuming. Knowledge from business processes to describe the tasks for achieving a goal, even if available, is hard for a novice designer or a user to understand.

Difficulty to link the services and execute a process. There are a large number of services available on the web. Locating a suitable set of services and linking the identified services are challenging even for the experienced developers. Current work in service flow identification [47, 117] does not help to identify tasks as those methods are solely based on input and output parameters of services. A user communicates with a task through a user interface (UI). Current approaches in the UI generation for web services [47, 63] are based on technical descriptions, and therefore, are difficult to understand and error prone.

In this chapter, we address the aforementioned challenges. The web contains a lot of well-written instructions (such as eHow [36] and Wikihow [133]) to teach people how to perform a process (such as how to buy a camera, how to make a restaurant reservation and how to buy a movie ticket). These instructions describe a series of steps to perform a task. Our approach understands written instructions and guides a user to complete a process by discovering and integrating different services. Our goal is to build a knowledge base for a process using text mining techniques that exploit the structure of the how-to instruction web pages. Our approach automatically identifies a task model from written instructions in the web pages. We use a task model to

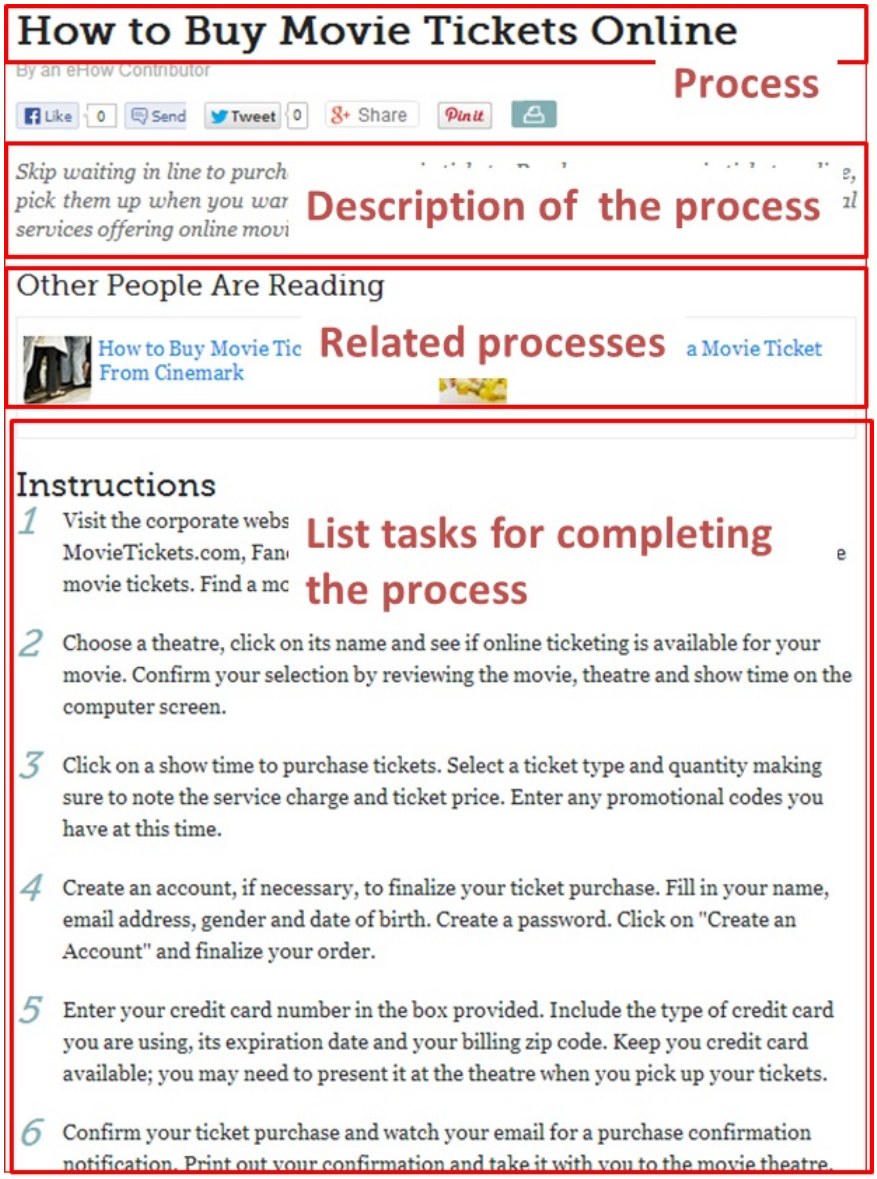


Figure 6.1: Annotated eHow article

identify and combine services to execute a process. To complete a process, a user needs a UI. Building a UI for a task is time consuming. We present an approach to generate a UI to execute a task. Our UI generation approach considers the data sharing between the services, so a user needs to provide minimum inputs for a task.

6.2 How-to Instruction Web Pages

How-to instructions in the web consist a knowledge base of human activities. These websites currently store millions of articles on how to do things step by step, which collectively cover almost every domain of lives, such as business, education, and travel. More or less all how-to instruction web pages have a similar format to present the content. Figure 6.1 shows an example of a web page with how-to instructions from an eHow website. The article describes the steps to buy a movie ticket online. Figure 6.1 contains four annotated parts, such as a process; a short description of the process; other related processes; and a list of tasks for completing the process. We observed three types of how-to instruction articles in the web. The first type of how-to instruction article helps a user to perform labor-intensive processes, such as how to clean a TV screen, and how to create a contact group on the iPhone. The second type contains a particularly descriptive and well-defined process, such as web pages related to recipes or hobbies. The third type describes the dynamic processes with many choices since the parameters and choices in a task vary from user to user. Examples of these tasks are reserving a seat in a restaurant, and going for a trip to Europe. Our approach cannot help with the first type of labor intensive processes. For the second and the third type of how-to instructions, we assist in finding the possible services and integrating the services to perform a process.

6.3 Task Model

A Task model describes the logical tasks that have to be carried out in a process to achieve a user's goals. Several task modeling notations exist, such as Hierarchical Task Analysis (HTA), Goals, Operators, Methods, Selection rules (GOMS), and

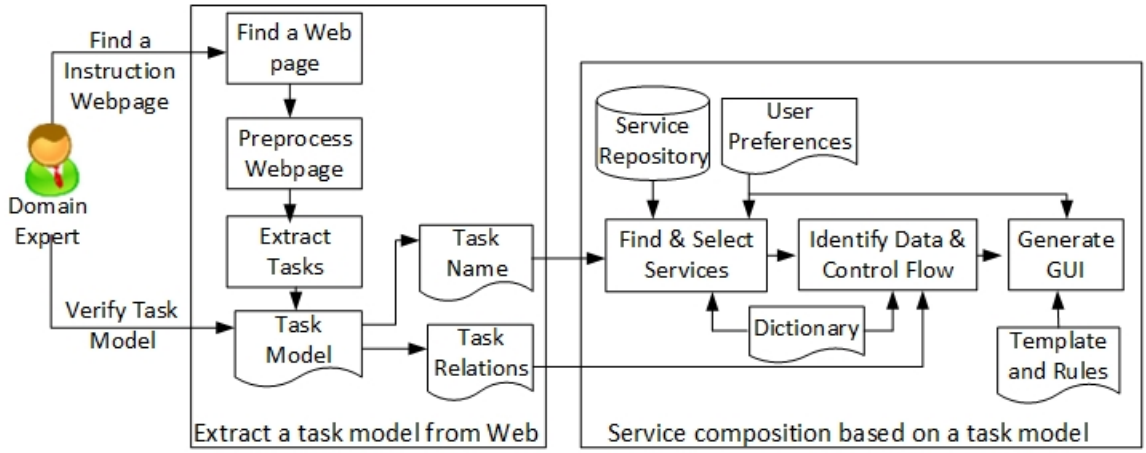


Figure 6.2: Overall steps to generate UI for a task from web services

ConcurTaskTrees (CTT). We use CTT to model tasks as CTT follows an engineering approach to task models. Moreover, the semantics of CTT are more complete and precise. It can support automated service discovery more effectively than other user task modeling formalism. CTT is applied in the field of end user programming. It is easy for end users to express and understand CTT. CTT [30] is a graphical notation that supports the hierarchical structure of tasks, which can be interrelated through a set of operators (such as iteration and optional) that describe the temporal relationships (such as choice, interleaving and enabling) between sub-tasks. CTT describes four types of tasks: the user, the application, their interaction and the abstract tasks. W3C specification for CTT [30] provides complete information on CTT Meta-model and relations among the tasks.

6.4 Overview of Our Approach

A user selects one of many how-to instruction web pages related to his goal. Our approach is to search and compose services based on a user selected web page. A

process is a well-defined, concrete action that a user performs to achieve a goal. In order to achieve a user's goal, we break a process into a set of tasks that can be scheduled and completed. Figure 6.2 shows the overall steps of our approach. We identify task models from how-to instructions in the web. We use the task models to find and compose services.

6.4.1 Task Model Extraction from Web Pages

In this sub-section, we introduce an approach that automatically extracts a task model from a semi structured web page.

Finding Web Pages

In this research, we examine two specific web sites to extract human written instructions (eHow [36] and Wikihow [133]). If a web page matches the user's scenario, we use this web page to extract a task model to form service composition.

Pre-processing How-to Instruction Web Pages

To analyze an HTML web page, we parse it to build a Document Object Model (DOM) tree structure. An HTML file may contain mismatched HTML tags, although it can be properly displayed by web browsers due to the fault-tolerant capability of web browsers. We use HTML syntax checker [60] to correct the malformed HTML tags. Then we parse the HTML into DOM tree structure. The pre-processor contains two steps: 1) learn the rules from the sample web pages; and 2) apply the rules on a web page to extract tasks. We manually examine and learn the DOM structure of the title and the instruction steps. We use these learned DOM structures to form

```
Input: InstructionSteps
Output: Action Object Lists
Algorithm
1. For each step in InstructionSteps
2. step=Instructionsteps[index]
3. List<Segment> seglist=ComputerSegment(step);
4.      For all Seglist sl∈seglists do
5.              ASegment=POSTagger(sl);
6.              List<ActionObject> aolist=ExtractActionObject(ASegment)
7.              List conjection=ExtractConjuction(Asegment)
8.              Aobj=Order(aolist, conj)
9.      End For
10. End For
```

Figure 6.3: Algorithm for identifying tasks from how-to instruction web pages

the rules. This procedure is based on the assumption that the documents collected from the same source share the common structures. The preprocessor uses wrapper induction approaches to extract the title and the instruction steps from a web page. The title becomes the root of a task model and the instruction steps become the tasks to complete a process (*i.e.*, extracting tasks).

Extracting Tasks

Each instruction step describes the tasks in a process. We extract the functional semantics of an instruction step. Functional semantics express the primary intent of a sentence in terms of the action-object pairs. An action-object pairing articulates what action is performed with an object. An action is represented by a verb and objects are described by nouns in a sentence. For example, in a sentence "buy a ticket", the functional semantic pair is {buy, ticket}. "Buy" and "ticket" correspond to the action and the object respectively. Our objective is to obtain tasks involved in

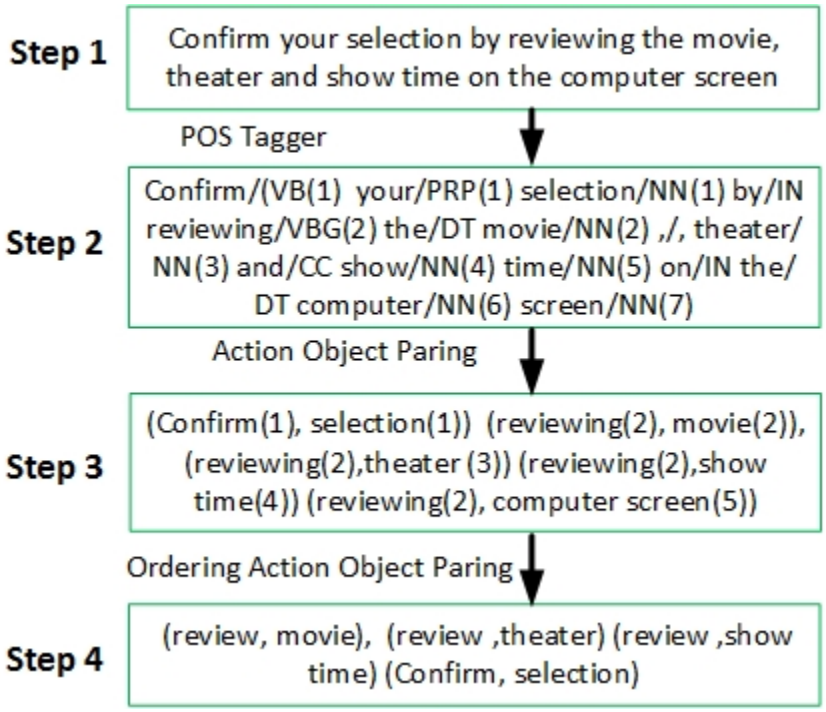


Figure 6.4: Example showing task extraction steps

a process (*i.e.*, instruction steps) as action-object pairs. We analyze the instruction steps to extract tasks as functional semantics. Figure 6.3 shows the algorithm to identify tasks. Instruction steps may contain multiple segments. A segment is a sentence in the instruction step containing one or more verbs and nouns. Line 3 in Figure 6.3 extracts the segments from an instruction step.

We use a well-stabilized part-of-speech (POS) tagger [120] to identify the syntactic structure of a segment. Step 2 in Figure 6.4 shows a POS tagged sentence. Each number (*i.e.*, inside parenthesis) in Step 2 indicates the index of the corresponding word as a noun or a verb in a sentence. For example, in Step 2 of Figure 6.4, selection is the first noun (NN) and the movie is the second noun (NN). We post-process the generated data structures to resolve object names consisting of multiple words (*e.g.*,

"Computer screen"), phrasal verbs (*e.g.*, "go to"), and pronominal referrals (*e.g.*, "it"). We assume "it" always refers to the last mentioned object, which is a sensible heuristic in most of the cases. For each segment, we extract actionable verb (VB)/verb phrases (VP) and the related noun phrases (NP). (VB/VP) or (NP) forms a task. Step 3 shows action-object pairs, such as "confirm selection", "reviewing movie", "reviewing theater", "reviewing showtime" and "reviewing computer screen".

We filter irrelevant tasks from the list of tasks extracted. PMI (Point-wise Mutual Information) [121] helps determine the relation between two words or phrases based on the information available in the web. We use Google Services [49] to calculate PMI. The PMI score is the number of hits for a query that combines domain and task divided by the hits for the task alone. This can be viewed as the probability that a domain and a task can be found on a web page as shown in 6.1.

$$PMI(domain, task) = \frac{hits(domain \text{ AND } task)}{hits(task)} \tag{6.1}$$

where hits (x) is the number of results that a search engine returns for a query x.

For example, in Step 4, we compute PMI for each term with the domain "movie" and each object in action-object pairs. As a result, $PMI(movie, reviewtheater)$ is 0.287; $PMI(movie, confirmselection)$ is 0.197; and $PMI(movie, reviewcomputerscreen)$ is 2.13415×10^{-5}. To identify the threshold to filter irrelevant tasks, we randomly selected 20 how-to instruction web pages and manually check the result PMI. We empirically set 10^{-3} as the threshold. In line number 6 of Figure 6.3, we compute the score. If the score is above the threshold, we consider it as an action-object pair. Hence, in Figure 6.4, (review, computer screen) is filtered out. We convert the verb to the base form. For the extracted pairs, we analyze its occurrence index and

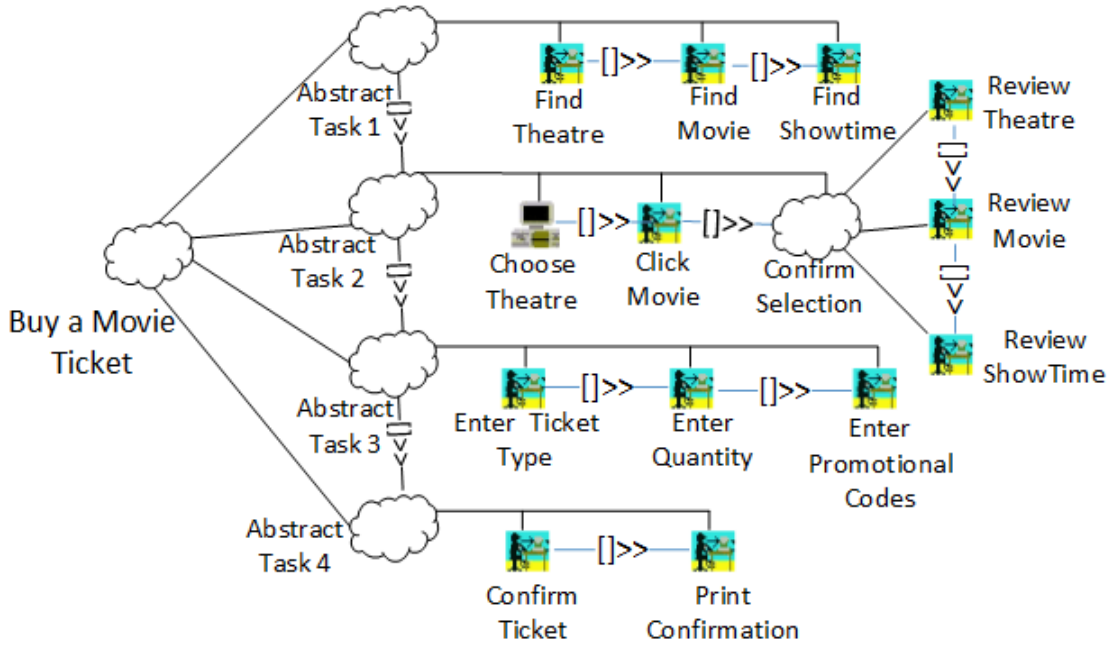


Figure 6.5: Simplified task model extracted from Figure 6.1

compare with clauses, such as "before", "after", and "by", to identify the order of the sequence. We further arranged the remaining action-object pairs {review movie, review theater and review selection } based on the hierarchical relationship among the words. In case of no relation, we order words according to their occurrences in a sentence.

Using the action-object extracted from Figure 6.3, we build a CTT task model. Figure 6.5 shows a task model to buy a movie ticket online. Only two kinds of temporal relationship exist in how-to instructions. If the two noun phrases are connected through "and" clauses, there is a sequential enabling info (*i.e.*, [ ] >>) relation. If the noun phrases are connected with "or" clauses there is a Choice (*i.e.*, [ ]). If the same task appears multiple times in a process, we choose the last place where the task had appeared and remove other occurrences. If a step contains multiple tasks,

we make an abstract task which connects all the tasks of a step. A user confirms the extracted model.

6.4.2 Service Composition based on the Task Model

In this sub-section, we present our approach to find and compose services based on the task model generated in Section 6.3.

Finding Services

In this step, we find the services for each step in the CTT task model. We use our work in concept-based service discovery to discover services. A concept is a semantic notion or a keyword for describing a subject, *e.g.*, "traveling" or "taxi reservation". Service repository indexes services based on concepts available in the service description documents as described in Chapter 4. While searching for a service, we perform the entity identification [97] and change the query if the services are not found. For example, if our service repository does not contain a service for "Buy a camera" then we change the query to a more general upper level domain "Buy a product". We use WordNet [86] as the knowledge base for transformation. For the task model shown in Figure 6.4, we identify the concepts to search for services. Table 6.1 lists the concepts extracted from the task model.

Identifying Control and Data Flows

We consider the task relations from the CTT model as a control flow of a composite service. We find the data dependency between the services and change the control flow based on the data dependency. A data dependency graph depicts the collaborative

Table 6.1: Different information extracted from a task model

Field	Heuristic and explanation	Example
Domain	Noun in the process title	In "How to Buy Movie Tickets Online", movie is the domain of process title.
Service Name	For each task, the first word corresponds a service name.	In "Review movies", review is related to the service name
Input	Noun occurring with the UI related words (Input, Enter, Fill, Click, Submit)	In "Enter Name", name is the word related to input
Output	Noun occurring with the UI related words (show, select, read, confirm, validate, check, review, decide, ensure, choose)	In "Show movie list", movie list is the word related to the output.

relations between the services related to different tasks. Each service has a name and takes input parameters and gives an output. Either the input or the output of a service can be empty, but, not both. Multiple input and output messages in a composite service are merged into a set of inputs or outputs. We exclude fault messages as they seldom contribute to the data flow to the subsequent services. Similarly, we exclude access keys for services as they do not contribute to the data flow.

The name of a service and the input/output parameters follow the conventions used in programming languages. Table 6.2 shows the rules to decompose input parameters and output parameters. After decomposing words, we use porter stemmer [99], which is the process for reducing derived words to their stem, base, or root form. For example, the words "fishing", "fished", "fish", and "fisher" have the same root word, "fish". For each word, we calculate the semantic similarity which is defined in Equation 6.2.

Table 6.2: Rules to decompose words

Rule	Before applying the rule	After applying the rule
CaseChange	FindCity	Find, city
	getMovie	Get, Movie
Sufx containing No.	City1	City
Underscore separator	Customer_Information	Customer, Information
Dash separator	Find-city	Find, City

$$Semantic(p_{s1}, p_{s2}) = \begin{cases} 1 & \text{if } p_{s1} \text{ and } p_{s2} \text{ are identical, or synonymous} \\ \frac{1}{\#\text{ links}} & \text{if } p_{s1}\text{,\& } p_{s2} \text{ have hierarchical relation} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

where, ps1 and ps2 are the name of parameters of services s1 and s2; link is the number of nodes to reach a common parent from the names of ps1 and ps2 in WordNet

WordNet helps to identify if two terms are semantically similar and to what degree they are similar. When the words are identical, or synonymous, the semantic similarity is 1. If there is a hierarchical relation, it depends on their similarity degree. If the semantic (p_{s1}, p_{s2}) is greater than a threshold (*i.e.*, 0.3), we consider it as a match. We choose a threshold by analyzing different service input and output parameters. For each pair of services, we evaluate the semantic similarity between the input and output parameters. Based on the similarity between parameters, we identify the linkage between different services.

Composing Services

To compare the functional similarity of two operations of web services, we examine the input parameters, the output parameters, and the name of a service. The name of a service (*e.g.*, "*getWeatherInformation*") is generally the concatenation of words which declare the functionality of the operation. The operation name in the service description may follow different conventions and is generally a compound word (*e.g.*, "*findCity*"). The rules to decompose the names are given in Table 6.2. After decomposing the words, we use Porter stemmer [99] which is the process for reducing inflected (or derived) words to their stem, base or root form. For the set of words used in the operations, we identify the semantic similarity between the words appearing in the names of operations using WordNet [86]. If the words have hypernym and hyponym relations, we consider that they are the same. The similarity of the names of the two services is evaluated as given by Equation 6.3. It measures the ratio between the number of common words in the operation names and the total number of words in both operation names. The value of service similarity (*i.e.*, SS) is 1 in case the names of the operations have the exact match and 0 when there is no match.

$$SS(s1, s2) = \frac{|words(s1) \cap words(s2)|}{|words(s1) \cup words(s2)|} \tag{6.3}$$

where, SS refers to the service similarity; s1 and s2 represent service names; words (s1) and words (s2) are the set of words appearing in the service names s1 and s2.

Input similarity is the ratio of the number of the same input parameters to the total number of the input parameters of two operations. The value of SI is 1 in case that the names of the operations have the exact match and 0 when there is no match. Similarly, the similarity between the output parameters is given in Equation 6.5. SO

is the ratio of the common types of the output parameters between the operations by the total number of output parameters in the operations. The value SO is 1 when the names of the operations have the exact match and 0 when there is no match.

$$SI(s1, s2) = \frac{|InputParameters(s1) \cap InputParamters(s2)|}{|InputParameters(s1) \cup InputParamters(s2)|} \tag{6.4}$$

where SI refers to the similarity in input parameters; s1 and s2 are the services; InputParameters (s1) and InputParameters (s2) refer to the names of input parameters in operations s1 and s2.

$$SO(s1, s2) = \frac{|OutputParameters(s1) \cap OutputParamters(s2)|}{|OutputParameters(s1) \cup OutputParamters(s2)|} \tag{6.5}$$

where, SO refers to the similarity in output parameters; s1 and s2 are the services; OutputParameters (op1) and OutputParameters (op2) refer to the names of output parameters in services s1 and s2.

The overall functional similarity between the two operations is specified in Equation 6.6. The values for service similarity (SS), input parameter similarity (SI) and output parameter similarity (SO) fall within the range of 0 and 1. We normalize functional similarity shown in Equation 6.4 by dividing it by 3. Therefore, the result of the functional similarity is between 0 and 1. 1 indicates an exact match between the two operations and 0 shows no match.

$$FunctionalSimilarity(s1, s2) = \frac{|SS(s1, s2) + SI(s1, s2) + SO(s1, s2)|}{|3|} \tag{6.6}$$

where, SS refers to the service similarity between the services s1 and s2; SI refers

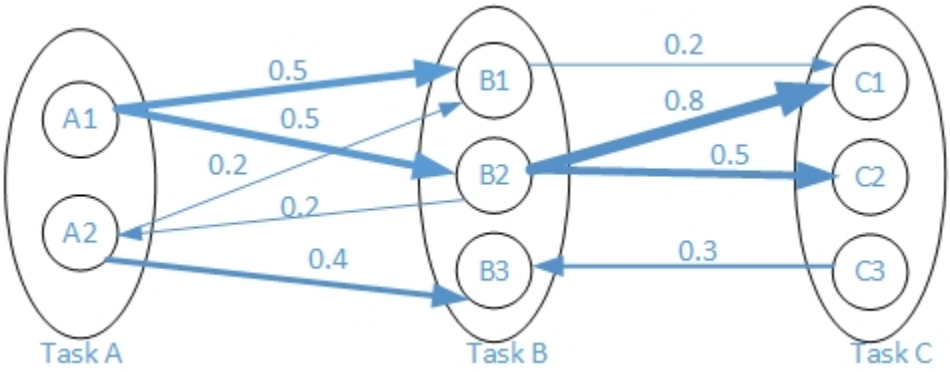


Figure 6.6: Dependency graph between different services in three different tasks

to the similarity in the input parameters of the operations s1 and s2; SO refers to the similarity in the output parameters of the operations s1 and s2.

For a set of services, if an output parameter of one service and an input parameter of another service are semantically similar, a data flow relation exists. We compute the semantic weight between the services which is the sum of semantic similarity between all parameters of two services. The semantic weight is normalized in interval [0, 1]. 0 means no data flow. 1 indicates that all the inputs of a service come from the output of another service. Figure 6.6 shows a simplified version of the dependency graph containing the services of different tasks. Moreover, the direction of the edges between two enclosed nodes dictates the data flow. The service where an arrow points takes the output of the preceding service as its input. For example, in Figure 6.6, service B2 of task B has two links between the services, C1 and C2 on task C.

We have a task model that defines the steps that a user needs to follow to perform a process. A task model helps us to find the relevant services and then gives a logical flow between the different services. However, the services perform a task. The execution order of the services can be different from the task model. Execution order also depends on the data dependency between the services. An executable task is a

graph G (V, E) where G is a Directed Acyclic Graph (DAG). Each vertex $vi \in V$ is a service. Each edge (u, v) represents a logical flow of messages from service u to service v. If there is an edge (u, v), then it means that an output message produced by service u is used to create an input message to service v. Our goal in this step is to combine one or more services in the dependency graph to form a task that can maximize the data sharing properties between the services. From Figure 6.6, we select A1 between {A1, A2} based on the semantic weight. We select B2 among {B1, B2} because its weight to C2 or C1 is the maximum. Hence, the flow will be A1→B2→C1.

The task model from some of the how-to instruction web pages, such as recipe web pages is different. Obviously, there are no services for the action-object pairs like "Boil Pasta". For such web pages, we define a service composition template and use a rule to invoke a predefined template. For example, for the recipe related web pages, we extract items and use invoke E-commerce templates to order items extracted from the recipe related how-to instruction web pages.

Generating User Interface

A user needs an interface to provide the data to perform a task. Each task may require more than one service. We want to increase the service-to-service interaction to minimize the demand for a user to enter information for accomplishing a task. Our service selection approach maximizes data sharing by choosing services with the maximum shared parameters. We extract all the parameters that may be required by the services in a process. We ask a user to fill all the parameters, avoiding multiple services-to-the user or user-to-service linkage.

We enhance Kasarda *et al.* [62] approach to generate UI when each service has

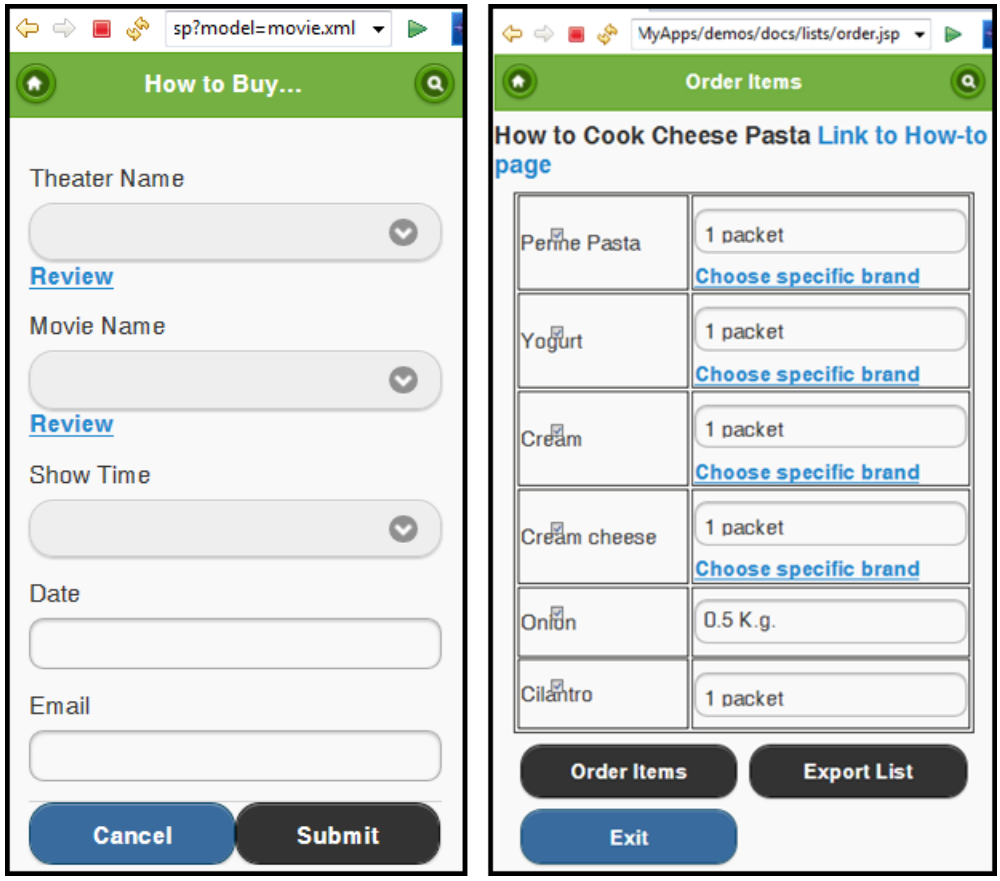


Figure 6.7: Screen-shot of generated user interface

different input and output parameters. The approach by Kasarda *et al.* is simple and easy to implement. The use of XHTML makes the generated UI adaptable to cross-platforms. The decision for the placement of inputs or outputs in the UI is based on the dependency between the services. The service input and output parameters are represented in XML. We enhance the approach by Kasarda *et al.* to find the relation among different input elements. Our enhancement also helps decide whether the input element should be user editable or not. We enhance the UI generation techniques using the following techniques:

1. If the output of a service is not used as an input parameter to another service,

the UI element of the output parameter is not user editable.

2. If the output of a service is a single parameter and used as an input parameter to another service, the UI element of the parameter is not user editable.

3. If the output of a service has multiple values (*i.e.*, array) and one of the elements is used as an input parameter to another service, the UI element of the parameter is user editable.

4. If the input parameter for a service does not come as an output from any other services, we select an appropriate UI element based on the approach described in [62].

5. If a node has multiple paths and if they do not merge to the same node later, the shortest path from the root becomes a new task and used as a link. For example, in Figure 6.7a, "review movie" and "review theater" are tasks used as links in "buy a ticket" process.

6. Based on the dependency found in section "Identify Control and Data Flow", we link service invocation with UI elements. Unless it is predefined in a template, we add two UI elements (submit and cancel), submit executes all the service invocations and the cancel exits the service composition. For example, as illustrated in Figure 6.7, the change in theater name alters the values in the selection box for the movie name which in turn modifies the values in the show time selection box.

UI elements for an executable task are dependent on each other. A change in one parameter can trigger a run-time change in another UI element. We also identify the

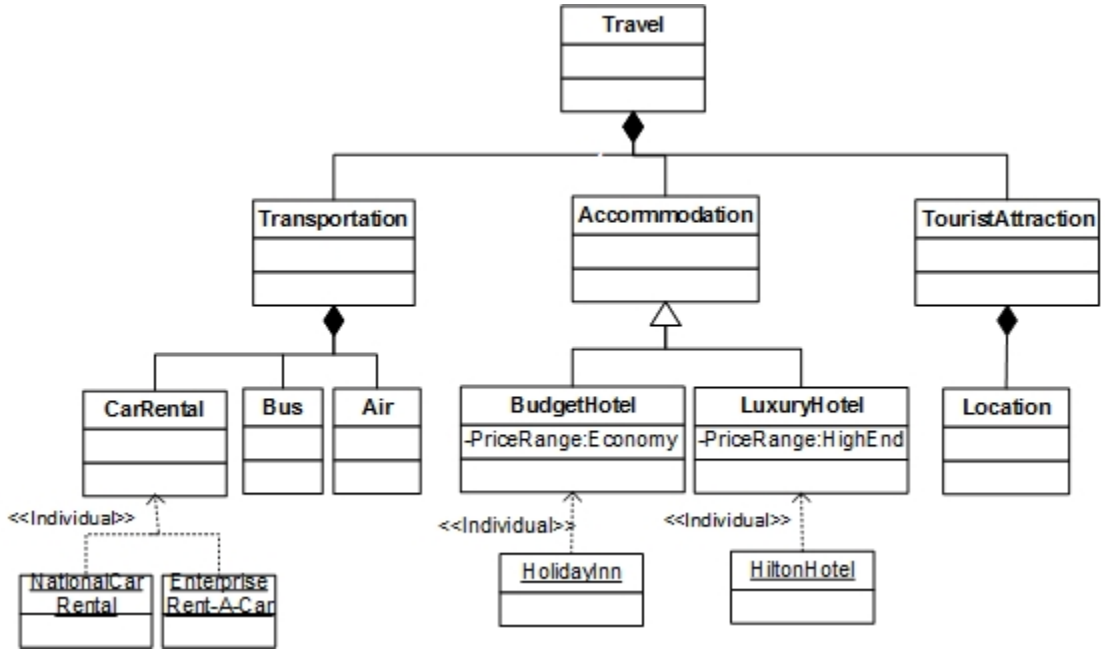


Figure 6.8: Example ontology

dependency between these parameters. Figure 6.7 shows the generated UI. Figure 6.7a shows a UI of a process to get a movie ticket. Figure 6.7b illustrates a UI to order items from a recipe based on a predefined template.

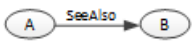
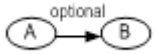
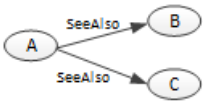
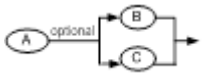
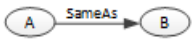
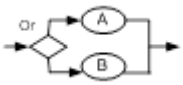
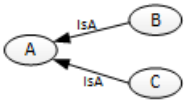
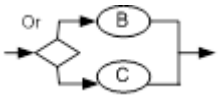
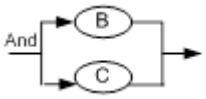
6.5 Linking and Executing Services

Our framework allows a user to specify a goal using a collection of keywords, and helps a user compose an ad-hoc process to fulfill the goal. We use the goal description to find a matching ontology (such as WordNet and ConceptNet) to extend the semantic meanings of the goal. An ontology represents knowledge as common entities (*e.g.*, people, travel, and weather), and the relations among the entities. In ontology, the semantics of a high-level goal is expanded into one or more concrete entities. Figure 6.8 illustrates an example ontology that defines "travel". The semantic of a high level

entity (*e.g.*, "travel") can be further expanded into four entities: "transportation", "accommodation", "tourist Attraction" and "car rental".

In our framework, the entities (*i.e.*, classes, individuals, and attributes) defined in the ontology are used to search for matching RESTful services from our concept-based service repository. However, a large number of resources could be returned and mixed together. It is a tedious job for users to manually select and organize RESTful services. To facilitate the selection of resources and identification of the interaction among the resources, our framework aggregates the functionally related resources into work items based on the structure of the matching ontology. To group the resources with similar functionalities, we identify the tasks for an ad-hoc process. For each task, we identify resources for a task based on the service discovery approach described in Chapter 4. We take an ontology which matches with the task description as the input. We use a step-wise approach to discover and organize the resources according to the level of abstraction. The high level entities in an ontology graph convey more abstract meanings suitable for discovering resources offering general purpose services. Such services allow users to receive the desired resources (*e.g.*, http://expedia.com) in one place without having to go through multiple servers for visiting different resources. For example, http://expedia.com provides a general service for planning a trip by providing information, such as car rental, flight ticket purchasing and hotel reservation. The low level entities in an ontology graph provide more specific meanings of a task indicating the possibility to find concrete resources which provide more specialized services or information. For example, to check into a flight operated by "Air Canada", a user has to visit more specialized resources by going to "Air Canada" website to print their boarding passes and check the flight

Table 6.3: Infer work item relations from resource graphs

Relation in a resource graph	Control relations among tasks	Description
		"SeeAlso" in a resource graph is interpreted as an "optional" relation which recommends another task to a user.
		Two tasks followed by the "optional" relation indicate that both tasks can be recommended to users.
		"SameAs" means two similar resources; therefore, the user only needs to perform one of the tasks.
		Resources B and C are instances of resource A. Thus resources B and C are similar. A user only needs to choose one from resources B and C
		Resource A contains resources B and C. Therefore, to handle A, users might need to perform the work items related to both B and C.

status.

To satisfy a user with varying needs in different levels of specialization, we use a breadth-first search algorithm to scan the ontology graph. We identify the general purpose resources from the top of the graph and the specialized resources from the low level of entities in the graph. To identify the control relations between the tasks, our framework analyzes the relations of resources captured in a resource graph. Table 6.3

summarizes the mappings from resources in a resource graph to the control relations among the tasks. As listed in Table 6.3, the "SeeAlso" relation in the resource graph is mapped to an "optional" relation in the ad-hoc process which is used to recommend tasks to a user. The user has the option to perform it or ignore it. The "SameAs" relation in the resource graph indicates that two resources are equal. Therefore, we convert the "SameAs" relation into an "Or" relation of tasks. "IsA" relation shows that one resource is an instance of another and these instances have the same features. Therefore, the siblings of "IsA" relation in the resource graph are converted to "Or" relation of tasks. "Contains" relation indicates that one resource is composed of other resources. Therefore, the elements in a "Contains" relation in the resource graph is converted to "And" relation among tasks. We further segment tasks into a work item if a set of tasks is related to one transaction. The relation between the work items is inferred from the relations among the segment of tasks.

Figure 6.9 presents an example of mapping the relations defined in the resource graph to the relations of related work items in an ad-hoc process. In Figure 6.9, the resources "Hilton", "Holiday_Inn", "Flight", and "Restaurant" are converted to work items in the ad-hoc process. In this example, if we trace the entire resource graph, these work items can be implemented by several tasks which are associated with more specialized resources. As a proof of concept of our proposed framework, we have designed and developed a prototype. We use a Firefox plugin to record a user's web browsing history. Our prototype analyzes the browsing history and connects different resources visited by users to generate ad-hoc processes in order to automate the repetitive tasks. To collect the resources of interest, our prototype allows a user to annotate the resources of interest using tags. Such resources are recorded and

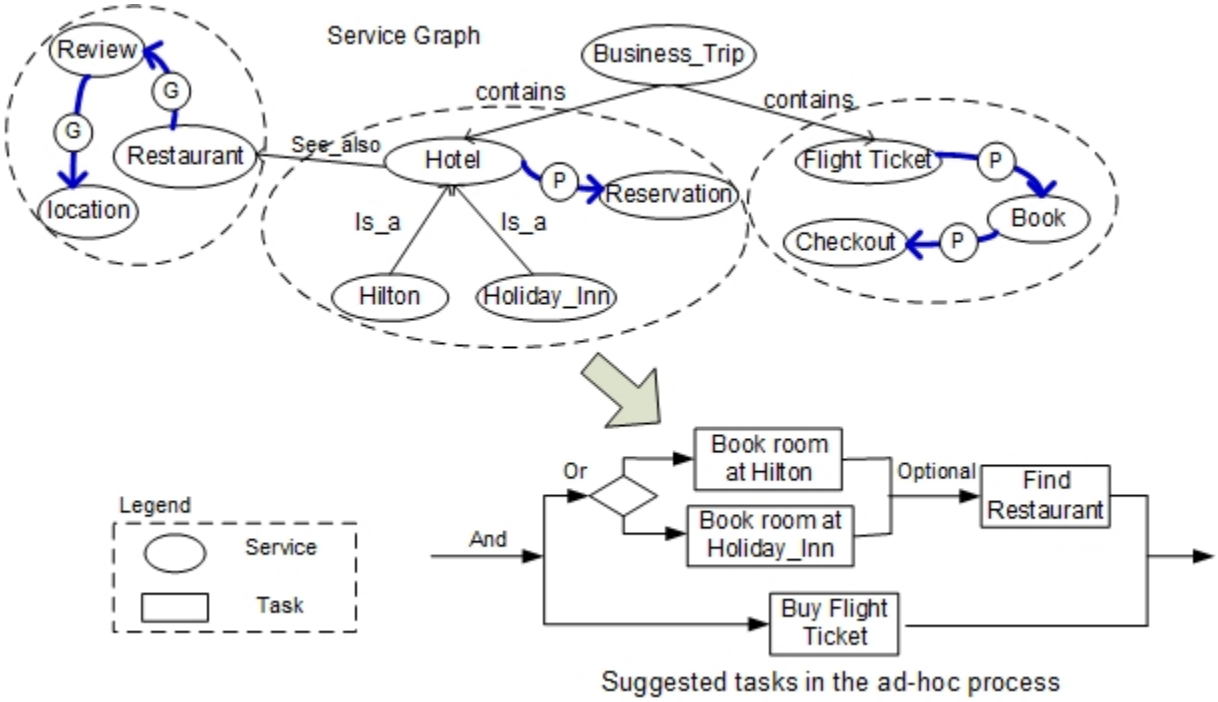


Figure 6.9: Example of a resource graph

converted to the unified schema.

Figure 6.10 shows the screen-shot of the prototype that helps a user compose resources. We implement our prototype using WireIT [132] utility which allows the user to wire different resources. The left hand side of our prototype illustrated in Figure 6.10 shows the resource interacted with the user. There are three sections on the right hand side of the screen depicted in Figure 6.10. The top section (*i.e.*, Tag Service) allows a user to tag and describe the functionality of resources as concepts. The middle one (*i.e.*, Personal Information) in the right hand side of the screen shown in Figure 6.10 describes the personal data, including address, credit card details, and online accounts. The bottom section (*i.e.*, utilities) contains the utilities that help the user invoke the defined ad-hoc processes following certain criterion, such as timing

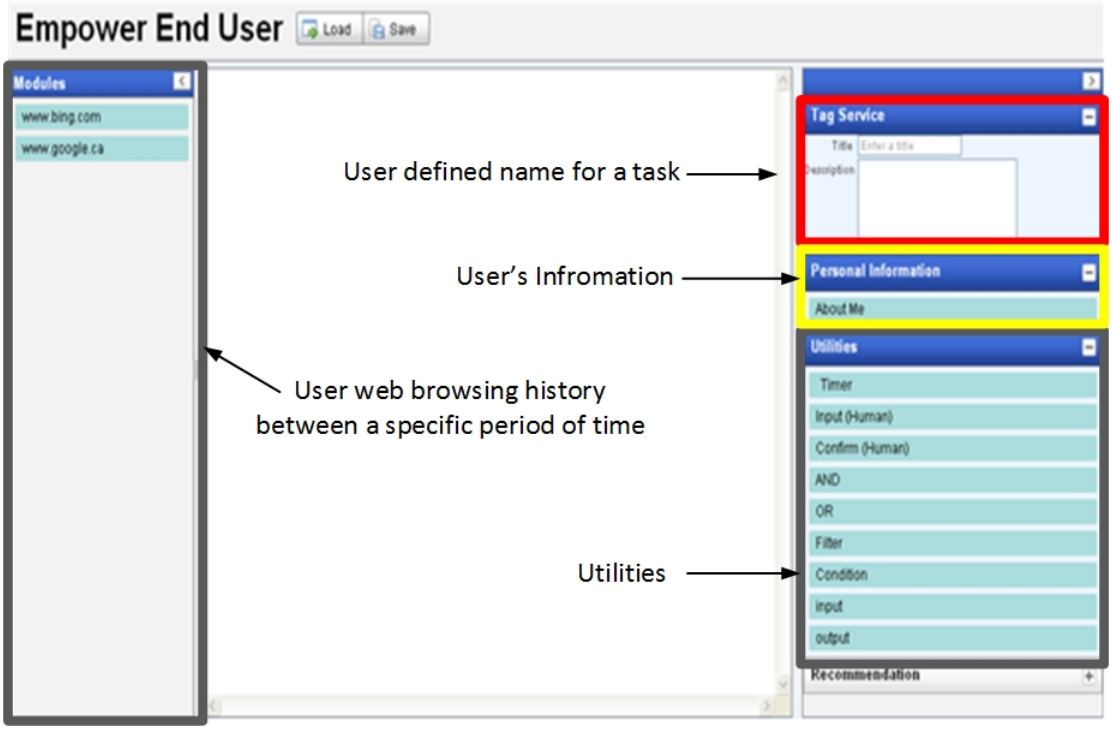


Figure 6.10: Annotated screen-shot of our prototype

constraints. The center region of Figure 6.10 enables a user to connect to resources.

Let us consider a use case scenario for our prototype. Assume that a user wants to attend a conference. The main activities involve booking a flight ticket and a hotel room. He may also need a taxi to transfer from the airport to the hotel. A user needs to enter flight information, such as "To (City, Country)", "From (City, Country)", and "Date". Similarly, a user needs the duration ("From Date" and "To Date") and the type of room as input to reserve a hotel room. The taxi reservation can leverage the input from the flight and hotel services. The time to rent a taxi is dependent on the time when a flight arrives. A user makes an ad-hoc process by connecting work items related to flight, hotel and taxi reservation. A user also connects different data types. Our approach selects the best services for each work item based on user

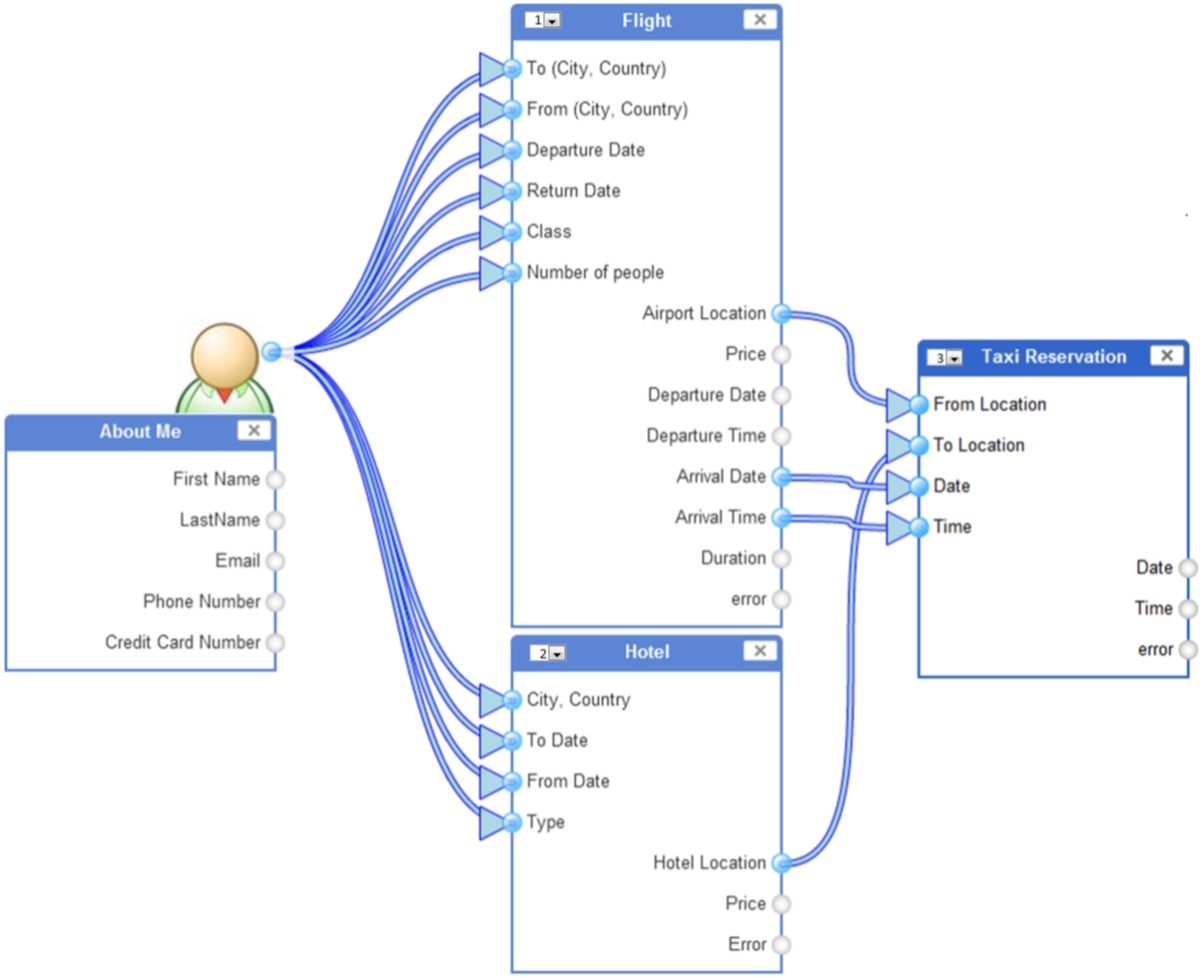


Figure 6.11: Annotated screen-shot for creating an ad-hoc process by a user

provided QoE attributes as described in Chapter 5. Figure 6.11 shows an ad-hoc process managing to travel to attend a conference.

6.6 Case Studies

We conduct a case study to evaluate the effectiveness of our approach. The objectives of our case study include: 1) evaluate the effectiveness to extract task models from how-to instruction web pages, and 2) evaluate the accuracy to compose services from

task models.

6.6.1 Setup

We collect 40 different how-to instruction web pages from eHow and Wikihow. The collected web pages were from different domains, such as communication (*e.g.*, "send SMS"), finance (*e.g.*, "find a stock price of a company"), and E-commerce (*e.g.*, "buy a product"). We avoid selecting many processes from the same domain to ensure the case study result is not skewed. In addition, we collect more than 600 service description files to examine service composition capability of our approach. The collected service description files have more than 4,000 different services. Our case study specifically finds effectiveness to extract a task model from web pages and accuracy of service composition based on task models.

6.6.2 Evaluation of Our Approach to Extract Task Models

We measure the effectiveness of our approach on identifying task models using precision and recall. As shown in Equation 6.7, the precision is the ratio of the total number of tasks correctly extracted to the total number of tasks in a how-to instruction web page. Recall, as shown in Equation 6.8, is the ratio of the total number of tasks correctly extracted to the total number of tasks existing in the how-to instruction web pages.

$$Precision = \frac{\{\text{relevant tasks}\} \cap \{\text{retrieved tasks}\}}{\{\text{retrieved tasks}\}} \tag{6.7}$$

$$Recall = \frac{\{\text{relevant tasks}\} \cap \{\text{retrieved tasks}\}}{\{\text{relevant tasks}\}} \tag{6.8}$$

Table 6.4: Results of our approach to extract task models

Domain	**Effectiveness of our approach to extract task model**		
	# Web pages	**Precision (%)**	**Recall (%)**
Hotel	10	91	57
Flight	10	89	53
E-commerce	8	92	61
Finance	7	88	58
Communication	5	93	70

Table 6.4 presents the effectiveness of our approach to extract task models from how-to instruction web pages. Table 6.4 shows the effectiveness of task identification of our approach. Our approach has the average precision of 90% and average recall of 59%. The reason for a lower recall is due to verb scoping during the task extraction step. When two verbs are conjoined, it is not clear whether the noun is associated with both or just the latter one. With "review and buy a camera" and "Go and buy a camera", for example, our approach needs to decide whether the noun (*i.e.*, camera) is associated with either both the verbs or just the one. Our approach could not correctly identify multi-word expressions (MWEs) in the instruction steps. MWEs represent the structure and meaning that cannot be derived from the component words as they occur independently. Examples of MWEs include conjunctions like 'as well as' (meaning 'including'), and phrasal verbs like 'find out' (meaning 'search').

6.6.3 Evaluation of Automatic Service Compositions based on Task models

We are interested in evaluating the accuracy of service composition based on the task model. Equation 6.9 gives the measure of accuracy. Accuracy is the ratio of the

Table 6.5: Accuracy to compose services from task models

Domain	**Accuracy of our approach to compose services**	
	# task Model	**Accuracy of SC (%)**
Hotel	3	88
Flight	2	85
E-commerce	4	83
Finance	3	85
Communication	3	95

correctly identified data and control flows by the total number of data and control flows among services.

$$Accuracy = \frac{\{\text{Correctly identified flow in SC}\}}{\{\text{Flow required by SC}\}} \tag{6.9}$$

To check the accuracy of service composition we first made sure that there are services available to perform the service composition for the task models. We selected the task model with at least one candidate service to form the service composition. For this case study, we selected 15 out of 40 task models. We manually verified the services, the data flows and control flows between the services selected.

Table 6.5 presents the result of our case study for automatic service composition based on a task model. Our approach has average accuracy of 85% to identify the correct flow between different services. The use of ambiguous words and the words not available in WordNet cause difficulties in identifying semantic similarity. Some element names were misspelled or inconsistently named. *e.g.*, an element "conferenceIdentifier" was misspelled as "conderenceIdentifier" in a conference management service. Hence, our approach could not identify the data flow between the services. Some of the entities are named differently, *e.g.*, "ASIN" and "OfferListingID" were

used interchangeably in Amazon Product API. We were unable to identify these interchangeable entities.

6.6.4 Threats to Validity

In this sub-section, we discuss the limitations of our approach and the threats that may affect the validity of the results of our case study. The manual verification introduces bias because a single evaluator could make mistakes. We should have recruited additional people for the evaluation. Unfortunately, we were not able to recruit more evaluators with sufficient knowledge about service-oriented applications and who could spend considerable time to manually inspect our results. To generalize our results for task extraction and service composition in other domains, we chose to study systems with a variety of domains to help ensure the generality of our results. Even though we still need to experiment with more domains and a wider variety of tasks from each domain.

6.7 Summary

This chapter presents an approach to build the task models from the how-to instruction web pages. Our case study shows that our approach has a high precision in identifying the instructions from how-to instruction web pages. In most cases, our approach can correctly identify the tasks. Similarly, we use the task model to find the relevant services to execute the tasks. Our approach can discover the data flows between the services. Given a correct task model, our approach can build an executable process with 90% accuracy. We believe the UI generation process is still a complex issue. The manual creation process is time consuming and complex because

it requires the combination of the work from the application developers and the UI designers. We designed and developed a tool that intends to ease and enhance the automatic UI generation process.

Chapter 7

Conclusions and Future Work

In this thesis, we present a framework to shield users from the complexity of SOA while taking the advantage of the benefits of SOA. Instead of requiring users to specify the detailed steps to achieve a goal, our approach extracts the process knowledge from the articles and information available on the web. We provide an approach to find appropriate services by indexing services based on the concepts available in the service description documents. Our approach helps users to formulate a web service search query by suggesting concepts and identifying the relations among the concepts. To facilitate the lack of QoS information for web services, our approach extracts QoE attributes from the web and ranks services based on these attributes. By automatically finding the data flow relations among the services, our approach links and executes services to achieve users' goal. Users play a central role in different stages of our approach. Whenever possible, we use user-generated content (*e.g.*, how-to instructions and reviews) to perform service composition.

7.1 Contributions

The major contributions of this thesis are summarized as follows.

Identifying RESTful services from heterogeneous services. Heterogeneity in services and service description documents makes service composition time consuming and difficult. We provide an approach to identify resources associated with each operation in SOAP-based services. Our approach also identifies reusable business functionality from web applications as RESTful services. To shield users from dealing with diverse service description languages, we propose a unified model to represent various kinds of services as RESTful services.

Extracting concepts for service discovery. The precision of service discovery approaches is dependent on the understanding of the semantics of service description documents and users' queries. The vocabulary adopted to describe a service usually contains technical words from the software development domain. Words in the service description documents can be different from the ones used in users' queries. We propose an approach to extract concepts from service description documents. We link concepts based on semantic relationships. We help to reduce the semantic gap between service providers and web service search queries. Our approach helps users to make their queries more specific, which in-turn increases the precision of the retrieved services.

Developing techniques to extract Quality of Experience (QoE) from reviews. QoS is a decisive factor to choose services from a pool of equivalent services. However, QoS information is difficult to find. Run-time monitoring to capture QoS information is resource-intensive. Instead of only relying on QoS, we use online reviews to rank and select services. We provide an approach to extract QoE attributes of a service from online reviews. Our approach is highly precise in extracting QoE

attributes. Correlations between QoE attributes and QoS attributes provide an opportunity to use QoE attributes for service selection, when QoS attributes are not available. Our approach allows users to specify their preferences as QoE attributes. Hence, the service selection and the execution path selection are conducted based user specified QoE attributes.

Extracting process knowledge from the web and using the knowledge to link and execute services. In the current state of practice, SOA practitioners manually describe the details of each task and the interactions among tasks using BPEL. Users may not have a clear plan to achieve a goal. We present an approach to extract process knowledge from how-to instruction web pages. Through experiments, we showed that our approach extracts task models from how-to instruction web pages with high precision. We derive resource graphs based on user's task description. Our approach helps users to generate ad-hoc processes from resource graphs. Different users have different requirements and preferences in the service composition results. We implement a framework that allows users to perform a service composition.

7.2 Future Work

This section describes our future research to solve the limitations that we have observed in our current approaches.

Acquiring process knowledge from other sources available in the web. In this thesis, we primarily use how-to instruction to obtain process knowledge. However, we can enhance our work to obtain process knowledge from other information sources such as emails and frequently asked questions.

Sharing and reusing data between different tasks. Automatic generation

of input data for tasks can reduce the workload of users and increase the automation of a task. In our current solution, users manually provide the input data for services. In the future, we plan to develop techniques to automatically generate service input data by analyzing users' context, historical execution of tasks, and task relations.

Reusing service composition results. Different users may pose similar requests during service composition. It is likely to improve the system efficiency if the solution of the new request can be adapted from previously generated service composition solutions. Collection of prior service compositions can be stored in the service repository. It is possible to generate implementation skeletons that can be refined into a full solution. However, this extension needs adaptation. For example, if some services in the previous solution are no longer available, the service composition needs to adapt the services to the available ones.

Bibliography

[1] P. Archer A. Perego, S. Konstantopoulos. Protocol for web description resources (POWDER): Powder-s vocabulary, January 2014. URL: `http://www.w3.org/2007/05/powder-s`.

[2] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. *Mining process models from workflow logs*. Springer, 1998.

[3] Marco Aiello, M Papazoglou, Jian Yang, M Carman, Marco Pistore, Luciano Serafini, and Paolo Traverso. A request language for web-services based on planning and constraint satisfaction. In *Technologies for E-Services*, pages 76–85. Springer, 2002.

[4] Eyhab Al-Masri and Qusay H Mahmoud. Investigating web services on the world wide web. In *Proceedings of the 17th International Conference on World Wide Web*, pages 795–804. ACM, 2008.

[5] Rosa Alarcón and Erik Wilde. RESTler: crawling restful services. In *Proceedings of the 19th International Conference on World wide web*, pages 1051–1052. ACM, 2010.

[6] Asil A Almonaies, Manar H Alalfi, James R Cordy, and Thomas R Dean. Towards a framework for migrating web applications to web services. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, pages 229–241. IBM Corp., 2011.

[7] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer Publishing Company, Incorporated, 2010.

[8] Areeb Alowisheq, David E Millard, and Thanassis Tiropanis. Express: Expressing restful semantic services using domain ontologies. In *The Semantic Web-ISWC 2009*, pages 941–948. Springer, 2009.

[9] José Luis Ambite, Genevieve Giuliano, Peter Gordon, Qisheng Pan, Naqeeb Abbasi, LanLan Wang, and Matthew Weathers. Argos: dynamic composition of web services for goods movement analysis and planning. In *Proceedings of the 2005 national Conference on Digital government research*, pages 275–276. Digital Government Society of North America, 2005.

[10] Alain Andrieux and Karl Czajkowski. Web services agreement specification (ws-agreement), 2004.

[11] Knarig Arabshian, Christian Dickmann, and Henning Schulzrinne. Ontology-based service discovery front-end interface for gloserv. In *The Semantic Web: Research and Applications*, pages 684–696. Springer, 2009.

[12] M. Athanasopoulos and Kostas Kontogiannis. Identification of REST-like resources from legacy service descriptions. In *Reverse Engineering (WCRE), 2010*

17th Working Conference on, pages 215–219, 2010. doi:10.1109/WCRE.2010.31.

[13] Wolf-Tilo Balke and Matthias Wagner. Towards personalized selection of web services. In *WWW (Alternate Paper Tracks)*, pages 20–24, 2003.

[14] Boualem Benatallah, Mohand-Said Hacid, Alain Leger, Christophe Rey, and Farouk Toumani. On automating web services discovery. *The VLDB Journal*, 14(1):84–96, 2005.

[15] Bing, January 2014. URL: http://www.bing.com/.

[16] M Brian Blake, Daniel R Kahan, and Michael Fitzgerald Nowlan. Context-aware agents for user-oriented web services discovery and execution. *Distributed and Parallel Databases*, 21(1):39–58, 2007.

[17] Phil Blanco, Rick Kotermanski, and Paulo Merson. Evaluating a service-oriented architecture. Technical report, DTIC Document, 2007.

[18] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[19] Tom Broens, Stanislav Pokraev, Marten Van Sinderen, Johan Koolwaaij, and Patricia Dockhorn Costa. Context-aware, ontology-based service discovery. In *Ambient Intelligence*, pages 72–83. Springer, 2004.

[20] Marcello Bruno, Gerardo Canfora, Massimiliano Di Penta, and Rita Scognamiglio. An approach to support web service classification and annotation. In *e-Technology, e-Commerce and e-Service, 2005. EEE'05. Proceedings. The 2005 IEEE International Conference on*, pages 138–143. IEEE, 2005.

[21] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, Francesco Perfetto, and Maria Luisa Villani. Service composition (re) binding driven by application–specific qos. In *Service-Oriented Computing–ICSOC 2006*, pages 141–152. Springer, 2006.

[22] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of the 2005 Conference on Genetic and evolutionary computation*, pages 1069–1075. ACM, 2005.

[23] Fabio Casati, Ski Ilnicki, LiJie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. Adaptive and dynamic service composition in eflow. In *Advanced Information Systems Engineering*, pages 13–31. Springer, 2000.

[24] Dipanjan Chakraborty, Filip Perich, Anupam Joshi, Timothy Finin, and Yelena Yesha. A reactive service composition architecture for pervasive computing environments. In *Mobile and Wireless Communications*, pages 53–60. Springer, 2003.

[25] Sirish Chandrasekaran, Sirish Ch, Samuel Madden, and Mihut Ionescu. Ninja paths: An architecture for composing services over wide area networks. Technical report, UC Berkeley, 2000.

[26] C-H Chang, Mohammed Kayed, R Girgis, and Khaled F Shaalan. A survey of web information extraction systems. *Knowledge and Data Engineering, IEEE Transactions on*, 18(10):1411–1428, 2006.

[27] Sheng-Tzong Cheng, Jian-Pei Liu, Jian-Lun Kao, and Chia-Mei Chen. A new framework for mobile web services. In *Applications and the Internet (SAINT) Workshops, 2002. Proceedings. 2002 Symposium on*, pages 218–222. IEEE, 2002.

[28] Uddam Chukmol. A framework for web service discovery: service's reuse, quality, evolution and user's data handling. In *Proceedings of the 2nd SIGMOD PhD workshop on Innovative database research*, pages 13–18. ACM, 2008.

[29] Philipp Cimiano and Johanna Völker. Text2onto. In *Natural Language Processing and Information Systems*, pages 227–238. Springer, 2005.

[30] Concur task trees (ctt), January 2014. URL: `http://www.w3.org/2012/02/ctt/`.

[31] Chiara Di Francescomarino, Alessandro Marchetto, and Paolo Tonella. Reverse engineering of business processes exposed as web applications. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*, pages 139–148. IEEE, 2009.

[32] Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Similarity search for web services. In *Proceedings of the Thirtieth International Conference on Very large data bases-Volume 30*, pages 372–383. VLDB Endowment, 2004.

[33] Schahram Dustdar and Wolfgang Schreiner. A survey on web services composition. *International Journal of Web and Grid Services*, 1(1):1–30, 2005.

[34] G. Meredith E. Christensen, F. Curbera and S. Weerawarana. Web services description language (WSDL) 1.1, January 2014. URL: `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`.

[35] Ebay, January 2014. URL: `http://www.ebay.ca/sch/i.html?_nkw=IPhone`.

[36] eHow — Discover the expert in you., January 2014. URL: `http://www.eHow.com`.

[37] Khalid Elgazzar, Ahmed E Hassan, and Patrick Martin. Clustering wsdl documents to bootstrap the discovery of web services. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 147–154. IEEE, 2010.

[38] Charles Engelke and Craig Fitzgerald. Replacing legacy web services with restful services. In *Proceedings of the First International Workshop on RESTful Design*, pages 27–30. ACM, 2010.

[39] Thomas Erl. *Service-Oriented Architecture (SOA) Concepts, Technology and Design*. Pearson Education India, 2005.

[40] Thomas Erl. *SOA design patterns*. Pearson Education, 2008.

[41] Andrea Esuli and Fabrizio Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *In Proceedings of the 5th Conference on Language Resources and Evaluation (LREC06*, 2006.

[42] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[43] Adobe - install adobe flash player, January 2014. URL: `http://get.adobe.com/flashplayer/`.

[44] E. W. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[45] John Gekas. Web service ranking in service networks. In *European Semantic Web Conference (ESWC 2006)*, 2006.

[46] John Gekas and Maria Fasli. Automatic web service composition using web connectivity analysis techniques. In *W3C Workshop on Frameworks for Semantics in Web Services*, pages 9–10, 2005.

[47] Çagdaş Evren Gerede, Richard Hull, Oscar H. Ibarra, and Jianwen Su. Automated composition of e-services: Lookaheads. In *Proceedings of the 2Nd International Conference on Service Oriented Computing*, ICSOC '04, pages 252–262, New York, NY, USA, 2004. ACM. URL: `http://doi.acm.org/10.1145/1035167.1035203`, `doi:10.1145/1035167.1035203`.

[48] iGoogle, April 2014. URL: `https://www.google.com/search?q=filetype:wsdl`.

[49] iGoogle, January 2014. URL: `http://www.google.com/ig`.

[50] Google accounts, January 2014. URL: `https://accounts.google.com/ServiceLogin`.

[51] Lars Grammel and Margaret-Anne Storey. An end user perspective on mashup makers. *University of Victoria Technical Report DCS-324-IR*, 2008.

[52] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. Soap version 1.2 part 1: Messaging framework. *W3C Working Draft, DevelopMentor, Sun, IBM, Canon, Microsoft*, 2002.

[53] Yanan Hao, Yanchun Zhang, and Jinli Cao. Wsxplorer: Searching for desired web services. In *Advanced Information Systems Engineering*, pages 173–187. Springer, 2007.

[54] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Applied Statistics*, pages 100–108, 1979.

[55] Andreas Hess and Nicholas Kushmerick. Automatically attaching semantic metadata to web services. In *IIWeb*, pages 111–116, 2003.

[56] Julia Hoxha and Sudhir Agarwal. Semi-automatic acquisition of semantic descriptions of processes in the web. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 1, pages 256–263. IEEE, 2010.

[57] Hypertext transfer protocol – http/1.1, January 2014. URL: `http://www.w3.org/Protocols/rfc2616/rfc2616.html`.

[58] H. Lausen J. Farrell. Semantic annotations for WSDL and xml schema, January 2014. URL: `http://www.w3.org/TR/sawsdl/`.

[59] H Janicke and M Solanki. Policy-driven service discovery. In *Proceedings of the 2nd European Young Researchers Workshop on Service Oriented Computing*, pages 56–62, 2007.

[60] Jtidy, January 2014. URL: `http://jtidy.sourceforge.net/`.

[61] A. Sheth K. Gomadam, A. Ranabahu. SA-REST: semantic annotation of web resources, W3C submission, January 2014. URL: `http://www.w3.org/Submission/SA-REST/`.

[62] Ján Kasarda, Martin Nečaskỳ, and Tomáš Bartoš. Generating xforms from an xml schema. In *Networked Digital Technologies*, pages 706–714. Springer, 2010.

[63] Michael Kassoff, Daishi Kato, and Waqar Mohsin. Creating guis for web services. *IEEE Internet Computing*, 7(5):66–73, 2003.

[64] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[65] Jacek Kopecky, Karthik Gomadam, and Tomas Vitvar. hrests: An html microformat for describing restful web services. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 619–625. IEEE, 2008.

[66] M. Laitkorpi, J. Koskinen, and T. Systa. A uml-based approach for abstracting application interfaces to rest-like services. In *Reverse Engineering, 2006. WCRE '06. 13th Working Conference on*, pages 134–146, 2006. `doi:10.1109/WCRE.2006.8`.

[67] Grace Lewis, Edwin Morris, and Dennis Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In *Software*

Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on, pages 9–pp. IEEE, 2006.

[68] Frank Leymann et al. Web services flow language (wsfl 1.0), 2001.

[69] Qianhui Althea Liang, J-Y Chung, Steven Miller, and Yang Ouyang. Service pattern discovery of web service mining in web service registry-repository. In *e-Business Engineering, 2006. ICEBE'06. IEEE International Conference on*, pages 286–293. IEEE, 2006.

[70] Link relations - iana, January 2014. URL: `www.iana.org/assignments/link-relations?`

[71] Fangfang Liu, Yuliang Shi, Jie Yu, Tianhong Wang, and Jingzhe Wu. Measuring similarity of web services based on WSDL. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 155–162. IEEE, 2010.

[72] Hugo Liu and Push Singh. Conceptneta practical commonsense reasoning tool-kit. *BT technology journal*, 22(4):211–226, 2004.

[73] Yaling Liu and Arvin Agah. Crawling and extracting process data from the web. In *Advanced Data Mining and Applications*, pages 545–552. Springer, 2009.

[74] Yaling Liu and Arvin Agah. A prototype process-based search engine. In *Semantic Computing, 2009. ICSC'09. IEEE International Conference on*, pages 481–486. IEEE, 2009.

[75] Yutu Liu, Anne H Ngu, and Liang Z Zeng. Qos computation and policing in dynamic web service selection. In *Proceedings of the 13th International World*

Wide Web Conference on Alternate track papers & posters, pages 66–73. ACM, 2004.

[76] John Makhoul, Francis Kubala, Richard Schwartz, Ralph Weischedel, et al. Performance measures for information extraction. In *Proceedings of DARPA Broadcast News Workshop*, pages 249–252, 1999.

[77] Umardand Shripad Manikrao and TV Prabhakar. Dynamic selection of web services with recommendation system. In *Next Generation Web Services Practices, 2005. NWeSP 2005. International Conference on*, pages 5–pp. IEEE, 2005.

[78] Ryusuke Masuoka, Bijan Parsia, and Yannis Labrou. Task computing–the semantic web meets pervasive computing. In *The Semantic Web-ISWC 2003*, pages 866–881. Springer, 2003.

[79] E Michael Maximilien and Munindar P Singh. A framework and ontology for dynamic web services selection. *Internet Computing, IEEE*, 8(5):84–93, 2004.

[80] E Michael Maximilien and Munindar P Singh. Toward autonomic web services trust and selection. In *Proceedings of the 2nd International Conference on Service oriented computing*, pages 212–221. ACM, 2004.

[81] Sheila A McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *Intelligent Systems, IEEE*, 16(2):46–53, 2001.

[82] Jan Mendling and Martin Müller. A comparison of bpml and bpel4ws. In *Berliner XML Tage*, volume 2003, pages 305–316, 2003.

[83] PD Michailidis and KG Margaritis. String matching algorithms: Survey and experimental results. In *International Journal of Computer Mathematics*, 2000.

[84] Microformats, January 2014. URL: `http://microformats.org/`.

[85] George A Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

[86] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[87] North american industry classification system, January 2014. URL: `http://www.census.gov/eos/www/naics/`.

[88] Sangyoon Oh, Hasan Bulut, Ahmet Uyar, Wenjun Wu, and Geoffrey Fox. Optimized communication using the soap infoset for mobile multimedia collaboration applications. In *Collaborative Technologies and Systems, 2005. Proceedings of the 2005 International Symposium on*, pages 32–39. IEEE, 2005.

[89] Sangyoon Oh and Geoffrey C Fox. Optimizing web service messaging performance in mobile computing. *Future Generation Computer Systems*, 23(4):623–632, 2007.

[90] Nicole Oldham, Christopher Thomas, Amit Sheth, and Kunal Verma. Meteor-s web service annotation framework with machine learning classification. In *Semantic Web Services and Web Process Composition*, pages 137–146. Springer, 2005.

[91] Owl-s: Semantic markup for web services, January 2014. URL: `http://www.w3.org/Submission/OWL-S/`.

[92] A. Perego P. Archer, K. Smith. Protocol for web description resources (powder), January 2014. URL: `http://www.w3.org/TR/powder-dr/`.

[93] Massimo Paolucci, Takahiro Kawamura, Terry R Payne, and Katia Sycara. Semantic matching of web services capabilities. In *The Semantic WebISWC 2002*, pages 333–347. Springer, 2002.

[94] Mike P Papazoglou. The challenges of service evolution. In *Advanced Information Systems Engineering*, pages 1–15. Springer, 2008.

[95] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big'web services: making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web*, pages 805–814. ACM, 2008.

[96] Dunlu Peng, Sheng Huang, Xiaoling Wang, and Aoying Zhou. Management and retrieval of web services based on formal concept analysis. In *Computer and Information Technology, 2005. CIT 2005. The Fifth International Conference on*, pages 269–275. IEEE, 2005.

[97] Thierry Poibeau and Leila Kosseim. Proper name extraction from non-journalistic texts. *Language and computers*, 37(1):144–157, 2001.

[98] Shankar R Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. Icrafter: A service framework for ubiquitous computing environments. In *Ubicomp 2001: Ubiquitous Computing*, pages 56–75. Springer, 2001.

[99] Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[100] ProgrammableWeb - mashups, apis, and the web as platform, January 2014. URL: `http://www.programmableweb.com/`.

[101] Bhaskaran Raman, Sharad Agarwal, Yan Chen, Matthew Caesar, Weidong Cui, Per Johansson, Kevin Lai, Tal Lavian, Sridhar Machiraju, Z Morley Mao, et al. The sahara model for service composition across multiple providers. In *Pervasive Computing*, pages 1–14. Springer, 2002.

[102] Shuping Ran. A model for web services discovery with qos. *ACM Sigecom exchanges*, 4(1):1–10, 2003.

[103] Resource oriented interface description and declaration language, January 2014. URL: `http://www.wst.univie.ac.at/communities/riddl/`.

[104] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[105] Atul Sajjanhar, Jingyu Hou, and Yanchun Zhang. Algorithm for web services matching. In *Advanced Web Technologies and Applications*, pages 665–670. Springer, 2004.

[106] Sap news desk, microsoft, ibm, sap to discontinue uddi web services registry effort, soa world magazine, January 2014. URL: `http://soa.sys-con.com/node/164624`.

[107] Seekda, January 2014. URL: `https://www.seekda.com/int/`.

[108] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. In *ACm SIGIR Forum*, volume 33, pages 6–12. ACM, 1999.

[109] Evren Sirin, James Hendler, and Bijan Parsia. Semi-automatic composition of web services using semantic descriptions. In *Web services: modeling, architecture and infrastructure workshop in ICEIS*, volume 2003, 2003.

[110] Microsoft silverlight, January 2014. URL: `http://www.microsoft.com/silverlight/`.

[111] H.M. Sneed and S.H. Sneed. Creating web services from legacy host programs. In *Web Site Evolution, 2003. Theme: Architecture. Proceedings. Fifth IEEE International Workshop on*, pages 59–65, 2003. `doi:10.1109/WSE.2003.1234009`.

[112] Spidermonkey mozilla, January 2014. URL: `https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey`.

[113] Amanda Spink, Dietmar Wolfram, Major BJ Jansen, and Tefko Saracevic. Searching the web: The public and their queries. *Journal of the American society for information science and technology*, 52(3):226–234, 2001.

[114] World wide web consortium (W3C). semantic web, January 2014. URL: `http://www.w3.org/2001/sw/`.

[115] Web service semantics wsdl-s, ibm research report., January 2014. URL: `http://domino.research.ibm.com/library/cyberdig.nsf/0/ef9fe52551fb21dc8525710d005a8480?OpenDocument`.

[116] M. Tatsubori and K. Takahashi. Decomposition and abstraction of web applications for web service extraction and composition. In *Web Services, 2006.*

ICWS '06. International Conference on, pages 859–868, 2006. doi:10.1109/ICWS.2006.49.

[117] Johnson P Thomas, Mathews Thomas, and George Ghinea. Modeling of web services flow. In *E-Commerce, 2003. CEC 2003. IEEE International Conference on*, pages 391–398. IEEE, 2003.

[118] Min Tian, Thiemo Voigt, Tomasz Naumowicz, Hartmut Ritter, and Jochen Schiller. Performance considerations for mobile web services. *Computer communications*, 27(11):1097–1105, 2004.

[119] T. o'reilly, rest vs. soap at amazon, January 2014. URL: http://www.oreillynet.com/pub/wlg/3005.

[120] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, NAACL '03, pages 173–180, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. URL: http://dx.doi.org/10.3115/1073445.1073478, doi:10.3115/1073445.1073478.

[121] P. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *Proc. of the 12th European Conference on Machine Lerning*, 2001.

[122] Peter D Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual*

meeting on association for computational linguistics, pages 417–424. Association for Computational Linguistics, 2002.

[123] Uddi consortium. uddi executive white paper, nov. 2001, January 2014. URL: `http://uddi.org/pubs/UDDI_Executive_White_Paper.pdf`.

[124] United nations standard products and services code, January 2014. URL: `http://www.unspsc.org/`.

[125] WMP Van der Aalst. Dont go with the flow: Web services composition standards exposed. *IEEE intelligent systems*, 18(1):72–76, 2003.

[126] Steve Vinoski. Restful web services development checklist. *Internet Computing, IEEE*, 12(6):96–95, 2008.

[127] Web application description language, January 2014. URL: `www.w3.org/Submission/wadl/?`

[128] Yiqiao Wang and Eleni Stroulia. Flexible interface matching for web-service discovery. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 147–156. IEEE, 2003.

[129] Weather forecast, January 2014. URL: `http://www.theweathernetwork.com/weather/caon0349`.

[130] API Web services, WSDL, SDK, XML, SOAP .NET Web applications, January 2014. URL: `http://www.webservicelist.com/`.

[131] WebserviceX.NET :: XML Web Services solution provider, January 2014. URL: `http://www.webservicex.net/`.

[132] Wireit, January 2014. URL: http://neyric.github.io/wireit/.

[133] wikihow-how to do anything, January 2014. URL: http://www.wikihow.com/Main-Page.

[134] Woogle, January 2014. URL: http://db.cs.washington.edu/woogle.html.

[135] Web resource description language, January 2014. URL: http://www.prescod.net/rest/wrdl/wrdl.html.

[136] Oasis consortium (2007) web services business process execution language,version 2.0, January 2014. URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[137] The web service modeling language WSML, January 2014. URL: http://www.wsmo.org/wsml/wsml-syntax.

[138] Web service modeling ontology (WSMO), W3C member submission, January 2014. URL: http://www.w3.org/Submission/WSMO/.

[139] Hua Xiao, Ying Zou, Joanna Ng, and Leho Nigul. An approach for context-aware service discovery and recommendation. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 163–170. IEEE, 2010.

[140] Hua Xiao, Ying Zou, Ran Tang, Joanna Ng, and Leho Nigul. An automatic approach for ontology-driven service composition. In *Service-Oriented Computing and Applications (SOCA), 2009 IEEE International Conference on*, pages 1–8. IEEE, 2009.

[141] XMethods, January 2014. URL: http://www.xmethods.net/?

[142] Jiuyun Xu, Kun Chen, and Stephan Reiff-Marganiec. Using markov decision process model with logic scoring of preference model to optimize htn web services composition. *International Journal of Web Services Research (IJWSR)*, 8(2):53–73, 2011.

[143] Yuhong Yan, Min Chen, and Yubin Yang. Anytime qos optimization over the plangraph for web service composition. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 1968–1975. ACM, 2012.

[144] Minoru Yoshida, Kentaro Torisawa, and Junichi Tsujii. Extracting ontologies from world wide web via html tables. In *Pacific Association for Computational Linguistics"*, 2001.

[145] Hong Qing Yu and Stephan Reiff-Marganiec. Non-functional property based service selection: A survey and classification of approaches. In *In: Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop co-located with The 6th IEEE European Conference on Web Services*, pages 12 – 14, 2008.

[146] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z Sheng. Quality driven web services composition. In *Proceedings of the 12th International Conference on World Wide Web*, pages 411–421. ACM, 2003.

[147] Farhana H Zulkernine and Patrick Martin. An adaptive and intelligent sla negotiation system for web services. *Services Computing, IEEE Transactions on*, 4(1):31–43, 2011.