

CONTEXT-AWARE NON-ELECTRONIC SERVICE
DISCOVERY AND COMPOSITION

YIFEI ZHANG

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF APPLIED SCIENCE IN SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

JULY 2015

© YIFEI ZHANG, 2015

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Yifei Zhang**

Entitled: **Context-aware Non-electronic Service Discovery and
Composition**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science in Software Engineering

complies with the regulations of this University and meets the accepted standards
with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. N. Tsantalos

_____ Examiner
Dr. D. Goswami

_____ Examiner
Dr. J. Paquet

_____ Supervisor
Dr. Y. Yan

Approved _____
Chair of Department or Graduate Program Director

_____ 20 _____
Dr. A. Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Context-aware Non-electronic Service Discovery and Composition

Yifei Zhang

In today's web, many web services are created and updated on the Internet. In many cases, a single service is not sufficient to respond to the user's request and often services should be combined through service composition to fulfill business goals. Service discovery and service composition can be highly compatible with context, *i.e.*, according to context information, *e.g.*, location, budget and time, services are chosen and composed. Moreover, we include non-electronic services, *e.g.*, restaurants, movie theaters shopping malls and so on, into service composition. Non-electronic services are rarely considered in existing service composition research, however are frequently used in people's daily life. In this thesis, we provide an approach for using contexts to discover and compose non-electronic services. We present a new context model which is to make it more suitable for service composition. This model is also able to handle both low level sensor data and high level data in predicated logic. Our service composition algorithm uses soft constraints, dissatisfaction of which causes a penalty instead of the fail of planning. With this feature, the service composition algorithm can give the user several "good enough" solutions, instead of null solution. Additionally, a replanning module is developed to refine the solution according to user's further adjustments of his or her requirements. As a motivating example, a web based Personal Entertainment Planner system is built.

Acknowledgments

I would like to express my gratitude and thanks to my supervisor Dr. Yuhong Yan for her help, guidance and supervision during the production of this thesis. It is impossible for me to complete this thesis without her brilliant ideas.

I also would like to take this opportunity to thank my friends who give me lots of support and help.

Finally, the deepest gratitude goes to my friend Ran Zheng. Thank you for the great inspiration and encouragement.

Contents

List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Problem and Motivation	1
1.2 Contributions	3
1.3 Organization of the Thesis	4
2 Background	5
2.1 Web Services	5
2.1.1 Simple Object Access Protocol (SOAP)	6
2.1.2 Web Services Description Language (WSDL)	6
2.1.3 Representational State Transfer (REST)	7
2.1.4 SOAP vs. REST	8
2.2 Electronic services and non-electronic services	9
2.3 Context Modelling and Computing	9
2.3.1 Context Modeling Approaches	9
2.3.2 Web Application and Contextual Awareness	19
2.4 Context-aware Web Service Discovery and Mashup	21
2.5 Service Composition	22
2.5.1 Definition of Service Composition	22
2.5.2 Approaches for Service Composition	23
2.5.3 Planning Problem	24
2.6 Information Aggregation	24
2.7 A Motivating Example: Personal Entertainment Planner	25

3	Models for Contexts and Services	27
3.1	Context Model	27
3.2	Service Model	31
3.3	Context Evaluation	32
3.4	Comparison with Existing Context Models	37
4	Service Discovery and Mashup	42
4.1	Service Discovery in Practice	43
4.2	Available Search Engine Services	44
4.2.1	RESTful Services	44
4.2.2	HTML Services and Google Show Time	50
4.3	Service Discovery and Mashup	52
5	Service Composition	56
5.1	Problem Description	56
5.2	Composition Algorithm	59
5.3	Re-planning	64
5.4	Limitation	65
6	Implementation of the Personal Entertainment Planner	67
6.1	Introduction	67
6.1.1	Goals and Objectives	67
6.1.2	Statement of Scope	67
6.1.3	Major Constraints	68
6.2	Design Considerations	68
6.3	System Architecture	68
6.4	User View Application	71
6.4.1	Design Constraints	71
6.4.2	Input UI	71
6.4.3	Result List UI	72
6.5	Service Discovery Application	73
6.5.1	Application Architecture	73
6.5.2	Service Discovery Implementation	75
6.6	Service Mashup Engine	77

6.6.1	Building Models	77
6.6.2	Service Mashup Engine Implementation	79
6.7	Service Composition	80
6.7.1	Building Models	80
6.7.2	Service Composition Implementation	83
6.8	Re-planning Implementation	87
6.9	Tests	88
6.9.1	Experiment Setup	88
6.9.2	Experiment Procedure	89
6.9.3	Results	90
7	Conclusion	93
	Bibliography	96

List of Figures

1	Representation of Concepts Defined by WSDL 1.1 and WSDL 2.0 Documents [Wik14e]	7
2	Printer Entity [SMLP02]	11
3	CSCP Profile Example [BHH04]	13
4	Modeling the Scenario [HIR02]	14
5	Context Model Showing the Derivation Dependencies [HIR02]	14
6	The GUIDE Object Model [CMD99]	16
7	Partial Definition of CONON Upper Ontology [WZGP04]	18
8	Partial Definition of a Specific Ontology for Home Domain [WZGP04]	18
9	Possible Relations Between Two Time Intervals [Wik13]	35
10	Diagram of Mashup	53
11	Service Discovery and Mashup	54
12	Re-planning Process	65
13	System Architecture	69
14	System Sequence Diagram	70
15	Input UI	72
16	Setting Time	72
17	Choosing Travel Area	73
18	Input UI (Mobile)	74
19	Choosing Travel Area (Mobile)	74
20	Result List UI (Desktop)	74
21	Restaurant Detail	75
22	Re-planning Part (Desktop)	76
23	Result List UI (Mobile)	76
24	Re-planning Part (Mobile)	76

25	Architecture of Service Discovery Application	77
26	RESTful Service	77
27	Service Mashup Engine Class Diagram	78
28	Service Mashup Engine Class Diagram	80
29	User Inputs Sample	84
30	Process for the Algorithm	86
31	Service Mashup Engine Class Diagram (Re-planning)	88
32	Results for Service Discovery	91

List of Tables

1	Attributes and their Values	11
2	Ontology for Context	29
3	Variables	40
4	A Context Example	40
5	Relation Between Preferences and Real (time) Cost	41
6	The Request and Response of a RESTful Service	45
7	Google Place API Parameters	46
8	Yelp API Parameters	47
9	Foursquare Search Venues API Parameters	48
10	YellowAPI's Places API Parameters	49
11	Google Maps Parameters	49
12	The Request and Response of a HTML Service	51
13	Definitions of ω_g Based on Preferences	58
14	Results for Service Composition	91
15	Results of Satisfaction Evaluation	92

Chapter 1

Introduction

1.1 Problem and Motivation

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network [W3C04]. Available Web services are posted across the Internet using a set of open standards such as SOAP [W3C07a], REST [Fie02], and WSDL [W3C07b]. With these open standards, Web services are evocable and interoperable. The goal of Services Computing is to enable IT services and computing technology to perform business services more efficiently and effectively [IEE03]. When a single service cannot perform a business service perfectly, a composite service to fulfil the functional requirements of the business service is necessary. Automated Service Composition (ASC) is the generation of a business process to complete functional goals that cannot be fulfilled by individual services. Automated Service Composition as an approach related to Web services has drawn a lot of attentions [RS05]. Nowadays, many researchers only focus on the electronic services, *i.e.*, automatic services that are provided by software systems. Nonetheless, many real business services in our life such as restaurants, movie theatres and retail stores, which are non-electronic services. The scope of service computing should cover all kinds of services, including electronic services and non-electronic services.

A smart mobile phone user is a very good example of a person who uses non-electronic services such as restaurants, retail stores etc., not only particular electronic services, for his/her daily life. On the other hand, the requirements of those users are highly context-based. Context information, *e.g.*, location, identity, and time,

could be used as a part of requirements for service discovery and service composition, in addition to other business goals. The recent adoption of support for HyperText Markup Language (HTML5) [W3C14b] and associated technologies such as AppCache, IndexedDB, and Geolocation in Web browsers now enables us to collect context information through Web browsers or mobile Web browsers.

The contextual information can be collected easily. The difficulty is how to use the context properly for doing non-electronic service composition and service discovery. The contextual information collected from users could include lower level sensor data (*e.g.*, geolocation data) and high level data in predicate logic (*e.g.*, whether a user has seen a movie). Therefore, we need to propose a context model which can handle these data well. Moreover, the formats of different types of non-electronic services are various. So, we also need to define a service model which can be used to represent the various non-electronic services. Additionally, some operations should be defined over contexts, so that the system states could be transferred. For discovering non-electronic services, UDDI [OAS07] is not an option because public UDDI servers are practically unavailable [EFKS10]. Because most of non-electronic services providers are common search engines and Restful data services. We need to find an efficient way to retrieve non-electronic services from querying online resources based on users' contextual information. It is also important for us to make the mechanism of service discovery be extensive and efficient, because we may add new types of services in the future. Currently, many people focus on doing context-aware non-electronic service discovery or integration, like Tripadvisor [Tri13] and Expedia [Exp13]. However, in some scenarios, people will need a sequence of non-electronic services (a composite service) with a clear timeline in a fixed order to achieve their target. At that time, it is necessary to do the context-aware non-electronic service composition. The composition algorithm should be able to adopt the context model and service model. Moreover, the composition algorithm should have the ability to handle services from different domains. The composition algorithm should not only generate several solutions (plans) but also guarantee the quality of those solutions (plans), *e.g.*, variety and satisfaction. Additionally, it is necessary to ensure that the algorithm should produce not only solutions that satisfy the user's requirements perfectly but also "good enough" solutions that satisfy the user's requirements partially. So, we need to add soft constraints in the algorithm.

1.2 Contributions

In this thesis, we research context aware non-electronic service discovery and composition. We build a personal entertainment planner as a prototype to demonstrate our procedure of context-aware non-electronic service discovery and service composition. This application is a Web application. It has two versions: the mobile version and the desktop version. Our main contributions in this paper include the following:

1. We propose a new context model. Firstly, we build an ontology for the context. A part of the ontology is domain independent, while the domain dependent ontology can be added easily. We present the ontology for non-electronic service attributes, and this ontology can be used for extension to different scenarios. The ontology is used to fix the meaning of the names we use in this thesis. Our context model is built on the top of the ontology and is scenario dependent. This model is able to handle both logic values and real values. We consider a context to be the description of the current circumstance of an entity or a person (current context) and their business goals (target context). If a context satisfies the business goals, the context is a target context.
2. We provide one way to discover and mashup the non-electronic services. The preconditions and the post conditions of the services can be modeled as contexts and constraints to decide the execution order of the services in a composite process. We propose a general and efficient method to collect different types of services.
3. We develop a service composition algorithm which features soft constraints. We use the values of the soft constraints to represent the satisfaction degree of a plan. Dissatisfaction of those constraints does not invalidate a plan, but devalue it. With this mechanism, the service composition algorithm can always give the user several “good enough” solutions, instead of none solution. We also add a feature to ensure the diversity of those solutions.
4. We develop one module for re-planning (doing service composition again based on a user’s context if a user is not satisfied with the original solution).

1.3 Organization of the Thesis

The structure of this thesis is as follows: the first chapter describes the introduction of the problem, the motivation and the contributions. After that, the second chapter presents an overview of Web services, context modeling, service discovery and composition. Later in this chapter, we introduce a motivating example. In the third chapter, we introduce the related definitions in the model for context representation. We discuss composed service discovery and mashup in the fourth chapter. The fifth chapter includes the details related to algorithm of Web service composition. The sixth chapter describes the implementation of the entertainment planner. In the last chapter, we conclude the thesis by summarizing the contributions and pointing out possible future work.

Chapter 2

Background

2.1 Web Services

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

The definition of Web service is given by the World Wide Web Consortium (W3C)[w3c04b].

The term Web services describes a standardized way to integrate Web-based applications by using technologies like XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. Those applications can perform a wide variety of tasks, which range from simple request to complex processes. The communication between two applications through messages by using Extensible Markup Language (XML). These messages follow the Simple Object Access Protocol (SOAP). Those applications can be described by adopting Web Service Description Language (WSDL). They can be accessed using standard Internet protocols[CW10].

2.1.1 Simple Object Access Protocol (SOAP)

SOAP, originally an acronym for Simple Object Access protocol, is a lightweight protocol for the exchange of information in a decentralized, distributed environment [BEK⁺00] It is an XML based protocol and uses XML to define its message format. For message negotiation and transmission, SOAP frequently relies on other application layer protocols, most notably Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP). This protocol has three parts: the envelope, encoding rules, and representation of procedure responses and calls. Due to the nature of using many accepted technologies, there are some obvious benefits to use SOAP:

1. **Platform independent and language independent:** SOAP uses XML to encapsulate its information other than encoding the information into platform relevant binary format. This means we can use any programming languages and platforms to send and receive the messages without special interpreting. However, SOAP uses the XML format which needs to be parsed and is lengthier too which makes SOAP slower than those binary format.
2. **Works well with firewalls:** SOAP runs over HTTP and uses HTTP port, which eliminates firewall problems. When using HTTP as the protocol binding, an RPC call maps naturally to an HTTP request and an RPC response maps to an HTTP response. Moreover, SOAP messages are XML data which can be seen as transparent to firewalls.

2.1.2 Web Services Description Language (WSDL)

Web services usually describe their functionality and invocation formats to users by employing a structured method (XML). The Web Services Description Language (WSDL) is an XML-based interface definition language that is used for describing the functionality offered by a Web service [Wik14e]. The WSDL provides an official description for Web services which can be parsed automatically. Every Web service is associated with one WSDL document in order to let other Web services and applications to understand what services that Web services provide and the method to invoke. The WSDL defines Web service as a collections of “ports” (WSDL 1.1) or “endpoints” (WSDL 2.0) (see Figure 1). A port is defined by assigning a network

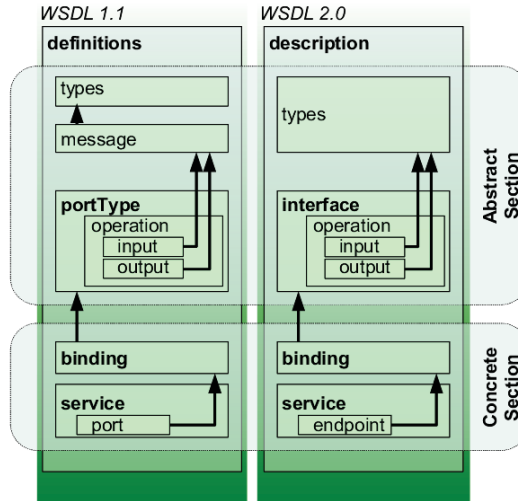


Figure 1: Representation of Concepts Defined by WSDL 1.1 and WSDL 2.0 Documents [Wik14e]

address with a reusable binding. Each “port” has a relevant “port type” (WSDL 1.1) or “interface” (WSDL 2.0). Those port types or interfaces are abstract collections of supported operations of Web services. Each “operation” usually has one “input message” and one “output message”. The format of “input message” and “output message” for an “operation” is defined at “message” sections and the data types of messages are defined in the “types” sections.

2.1.3 Representational State Transfer (REST)

Representational State Transfer (REST) has become more and more popular across the Web. It can be a simpler alternative to SOAP- and Web Services Description Language (WSDL)-based Web services [IBM08]. The RESTful style of Web services are based on REST which was firstly introduced in 2000 by Roy Fielding at the University of California, Irvine, in his academic dissertation [Fie00]. REST is a simple architecture that attempts to convert the interface or port to a set of uniform, standard operations (like GET, PUT, POST, DELETE). It concentrates on the communication with stateless resources rather than messages or operations. In the REST architectural style, the resources are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. So REST is able to communicate between clients written in different languages and focus on the resources. In fact, because of its usability, REST has

became a predominate Web service design model and has mostly displaced SOAP and WSDL-based interface design in many places [IBM08].

A RESTful Web service follows the four basic design principles listed below:

1. Uses HTTP methods explicitly;
2. Is stateless;
3. Indicates the directory structure-like URIs;
4. Transfers data using XML or Javascript Object Notation (JSON), or both.

2.1.4 SOAP vs. REST

Both SOAP and REST have their advantages and disadvantages. SOAP supports WS-Security which adds some enterprise security features. It supports communication through intermediaries, not just point to point. It also provides a standard implementation of data integrity and data privacy. SOAP also has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries. The drawbacks of SOAP include a heavy XML wrapper required for each request and response, difficulty in developing and tools needed for development. Depending on the features of SOAP, it is suitable when a formal contract must be established to describe the interface that the Web service offers and also when an application needs high safety level, like banking system.

On the other hand, REST also has its pros and cons. Its benefits include its ease for development; language and tool independence; better performance and scalability. The disadvantages of REST are that there is no common standard accepted yet for the formal REST service description; it does not have a complete support for security and REST requests cannot undertake large amount of data.

REST is recommended in the following situations [Ora06]:

1. Web services that are completely stateless;
2. The application can use caching infrastructures to leverage its performance;
3. The service producer and service consumer have a mutual understanding of the context and content being passed along;

4. Limited bandwidth;
5. Web services are needed to be delivered or aggregated into existing Web applications.

2.2 Electronic services and non-electronic services

In computer science, a service is a self-contained unit of functionality, such as retrieving an online bank statement. By that definition, a service is a discretely invocable operation. Based on the provider of services, we can classify services into two categories, electronic services and non-electronic services.

Electronic services are always generated by software systems, *e.g.*, Amazon EC2 service from Amazon Web Service. Most of electronic services have their inputs and outputs in predefined forms. The input of a electronic service could be the output of the other services and the output of a electronic service could also be the input of the other services. The key part in electronic service composition is to match the inputs and outputs for different electronic services.

Non-electronic services usually come from offline services or more traditional services, such as restaurants, movie theatres and shopping malls. In non-electronic services, they do not have explicit inputs and outputs. Each of non-electronic services may not have a clear dependency with the other services. When we do non-electronic service composition, we will focus on the properties of those services rather than relations between those services.

2.3 Context Modelling and Computing

2.3.1 Context Modeling Approaches

Currently, there is a great increase in research on how to use context awareness as an effective way to develop applications which are flexible, adaptable, and can react to a users' context without their intervention [BBH⁺10]. Several merits can be obtained by applying formal context aware information modeling. At first, the development of context aware applications requires adequate software engineering methods because of the complexity of those applications. The overall target is to make context aware

applications be evolvable. Therefore, the architecture of such applications should have an ability to be compatible with the change of the definition and evaluation of context information. A qualified context modeling formalism can push the complexity of context-aware applications into smaller space and reinforce their evolvability and maintainability. In addition, it is necessary to think about the re-use and sharing of context information between context-aware applications from the beginning, because the cost of collecting, evaluating and maintaining context information is very high [BBH⁺10]. High quality context information modeling will make the procedure of developing and deploying future applications easier. Moreover, for consistency checking, there is a demand for formally representing context information within a model [BBH⁺10].

Dey defines context as “any information that can be used to characterize the situation of an entity(user)” [Dey01]. Context modeling and computing provide a consolidated and efficient way to represent contexts and use contextual data. In this thesis, context is the information automatically detected by the Web browser (mobile or desktop), *e.g.*, the location, the identity, and the current time, or provided by the user *e.g.*, the budget and the range of moving that the user is willing to move around. In the existing work, there are several context modeling approaches which are classified by the schema of data structures [SLP04].

1. Key-Value Models

Key-value models use key-value pairs as a list of attributes and their values to model context information used by context-aware applications [BBH⁺10]. At the early phase, Schilit used key-value pairs to model the context by adopting the value of a context information, *e.g.*, location, to an application [SAW94]. Moreover, his work also investigates four types of context-aware applications using PARCTAB [AGS⁺93] [SAG⁺93]: proximate selection, automatic contextual reconfiguration, contextual information and commands, and context-triggered actions. These applications are able to adapt and be aware of the changes of context. Context information should include not only the user’s location but also the located-objects [SAW94]. Located-objects can be regarded as extensions of location information. There are three types of located-objects. The first kind is the input and output devices that require physical interaction, including printers, displays, video cameras and so on. The second kind includes objects

<i>Printer</i>
<i>Attributes</i>
<i>Colour = "yes"</i>
<i>Paper-size = "A4"</i>
<i>Duplex = "yes"</i>
<i>Technology = "laser"</i>
<i>Interface</i>

Figure 2: Printer Entity [SMLP02]

and services which are non-physical and can be accessed through particular locations; for example, bank accounts and menus. The third kind is the set of places users want to know including restaurants, night clubs and stores. Table 1 presents an example of these attributes and values of one context.

Table 1: Attributes and their Values

Attribute	Value
Location	In room 3172
Date and time	After October 1 between 10 and 12 noon

Context-Aware Packets Enabling Ubiquitous Service (CAPEUS) is a framework to realize context-aware selection and execution of services by using key-value models to describe the context [SMLP02]. In the framework, a user’s device should be regarded as an entry point for expressing service needs based on the changes of environment (context). [SMLP02] created a a uniform document format: Context-Aware Packets (CAPs), which is used for communicating service needs and expressing service needs on a high abstraction level. CAPs is composed of three parts: context constraints, scripting and data. The context constraints include *abstract entities*, *relations*, and *events*. Abstract entities are described by attributes which are a list of key-value pairs. Figure 2 shows a printer entity as an example. Generally speaking, a relation describes the dependencies of entities. One sample relation, *inRoom*, indicates that entities in this relation have to be at the same room. An event can be seen as a trigger to control the execution of a CAP initiated service call.

2. Markup Schema Models

Markup schema models are one type of context modeling approaches which use hierarchical data structures to describe contextual information [SLP04]. Those data structures are comprised of markup tags with attributes. The context of a markup tag is usually recursively defined by other markup tags. Markup schema models are used to represent profiles, *e.g.*, device capabilities. They often depend on a serialization of a derivative of *Standard Generic Markup Language* (SGML), which is the superclass of all markup languages such as the XML serializations. Composite Capabilities/Preferences Profile (CC/PP) [W3C14a] and User Agent Profile (UAProf) [WAP14] can be defined as extensions of SGML, which are expressed by RDF/S and a XML serialization. In order to cover the higher dynamics and complexity of contextual information, markup schema modeling methods often extend and complete the basic CC/PP and UAProf vocabulary and procedures [SLP04].

Comprehensive Structured Context Profiles (CSCP) are an example of this approach [BHH04]. CSCP is also based on RDF [W3C14c]. However, unlike CC/PP, CSCP does not have any fixed hierarchy. It prefers to express natural structures of profile information as required for context information rather than supports the full flexibility of RDF/S. Attribute names are interpreted context-sensitively according to their position in the profile structure. Therefore, unambiguous attribute naming throughout the whole profile is not required, which is needed for CC/PP. Another difference between CSCP and CC/PP is overriding mechanism. CC/PP provides for overriding of default attribute values only, CSCP supplies a more flexible overriding and merging mechanism, allowing for instance to override and/or merge a whole profile subtree. Figure 2 shows a CSCP profile example. *Pervasive Profile Description Language (PPDL)* is another context modeling approach in the markup scheme category which is not defined in the same way as CC/PP [CF03]. When interaction patterns have to be defined on a limited scale, PPDL has an ability to account for contextual information and dependencies. The relationship between the number of contextual aspects and the integration of the languages itself is uncomplicated. Due to the fact that no design criteria and only parts of the language are available to the public, the actual appropriateness of this context modeling approach is still unknown.

```

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:cscp="context-aware.org/CSCP/CSCPProfileSyntax#"
  xmlns:dev="context-aware.org/CSCP/DeviceProfileSyntax#"
  xmlns:net="context-aware.org/CSCP/NetworkProfileSyntax#"
  xmlns="context-aware.org/CSCP/SessionProfileSyntax#"
  <SessionProfile rdf:ID="Session">
    <cscp:defaults rdf:resource=
      "http://localContext/CSCPProfile/previous#Session"/>
    <device><dev:DeviceProfile>
      <dev:hardware><dev:Hardware>
        <dev:memory>9216</dev:memory>
      </dev:Hardware></dev:hardware></dev:DeviceProfile>
    </device>
  </SessionProfile>
</rdf:RDF>

```

Figure 3: CSCP Profile Example [BHH04]

There are several other context modeling methods belonging to the markup scheme category like Centaurus Capability Markup Language (CCML) [KKC⁺01] and ConteXtML [Rya99]. Most of them are often either proper or restricted to a set of contextual aspects, or both.

3. Graphical Models

Unified Modeling Language (UML) can be used as a typical representative of graphical models to describe the context in graphical structure because UML has a strong graphical component (UML diagrams) which is also appropriate to model the context.

A case study of context-aware communication in [HIR02] has the typical work of using graphical models. In this sample, Bob and Alice plan to arrange a meeting through their communication agents. Thus, the communication agents act as intermediaries for communication between Bob and Alice and also provide suggestions to them. In this scenario, those agents will collect the context information about the participants and their communication devices and channels because they rely on them. In Figure 4, [HIR02] uses both the Entity Relationship model and the class diagrams of UML to model the context information of the scenario of the case study. The model in Figure 4 includes three entity types: person, communication devices and communication channels. Each entity type is linked with several attributes, *e.g.*, person is associated with *Name*, *Activity*, *Location Coordinates*. From the figure, we can see that associations exist not only between the entities and their attributes but also the entities. For instance, both *Person* and *Device* are linked to the attribute of *Location Coordinates*. A dependency is a special kind of relationship between

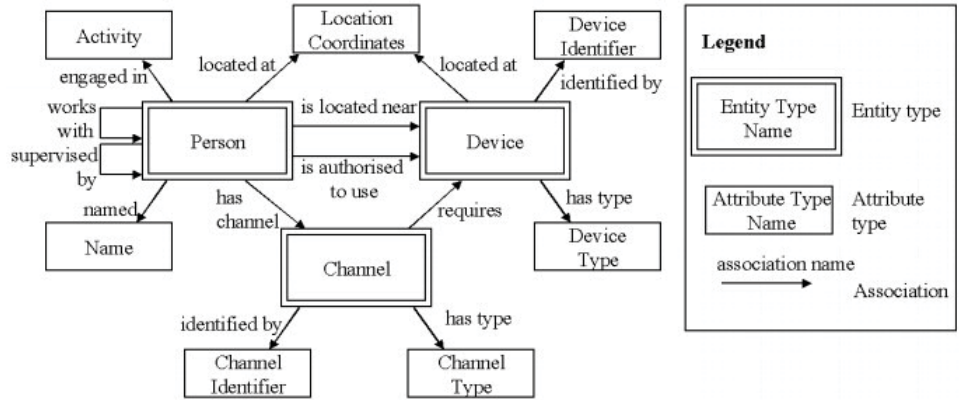


Figure 4: Modeling the Scenario [HIR02]

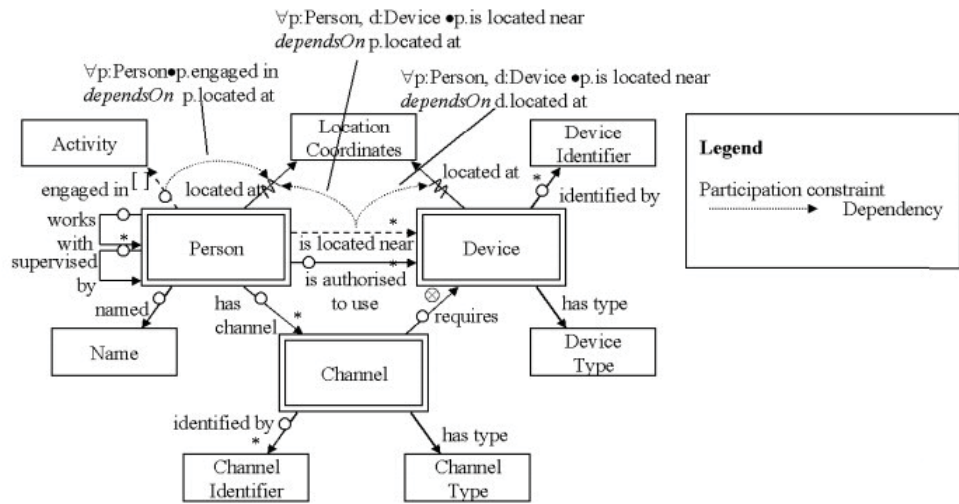


Figure 5: Context Model Showing the Derivation Dependencies [HIR02]

associations, which indicates the existence of a reliance of one association on another one [HIR02]. The derivation dependencies are shown in Figure 5 which presents the context model related to the case study. For example, the dependency $a_1 dependsOn a_2$ is described as a directed arc leading from a_1 to a_2 . [HIR02] is not the only research work using graphical models. Another sample is also a beautifully designed graphic oriented context model provided by Henricksen et al. [HIR03], which is an extension of the Object-Role Modeling (ORM) approach [Che76] which depends on some contextual classifications and description properties.

This type of approach is particularly suitable for deriving an ER-model and appropriate to build a relational database.

4. Object Oriented Models

Object oriented context modeling approaches are able to employ the main benefits of any object oriented approach, namely encapsulation and reusability. These merits can simplify the problems arising from the dynamics of the context. Because of the object oriented features, the details of context processing are encapsulated on an object level and the contextual information can only be accessed via specified interfaces.

A typical context-sensitive application using an object oriented model is the GUIDE project developed for portable Web-client-based machines (*e.g.*, tablet PCs and PDAs) [CMD99]. The GUIDE project is used to provide a tourist guide for visitors to the city and also recommend tours. Personal and environmental contextual information are two kinds of context-sensitive information which are modeled in this project. Personal information contains the visitor's interests, the visitor's current location, the time visitors plan to spend on their visit, their budget and any refreshment preferences they might have. An environmental context will include the time of the day, the weather, the season and the state of the city's transport system, *e.g.*, the location of traffic congestion or the closure of walkways. If a visitor has requested a tour of the city, then the GUIDE system should use both personal and environmental context to create a proper tour. Figure 6 shows the object models of the GUIDE. A visitor can interact with the GUIDE system over their own local Web browser like sending requests. From Figure 6, we can see that there are nine objects for the whole system. For instance, all HTTP requests are processed by the local Web server object, which may in turn need to interact with other objects in order to service the request. Each object is able to notify other objects by invoking its methods, *e.g.*, when a visitor enters a new or previously visited location, the *receiveNewPosition* method of local position object will be invoked in order to notify the control object.

Bouzy and Cazenave [BC97] also propose one way to adopt general object oriented mechanisms to represent contextual knowledge about temporal, goal,

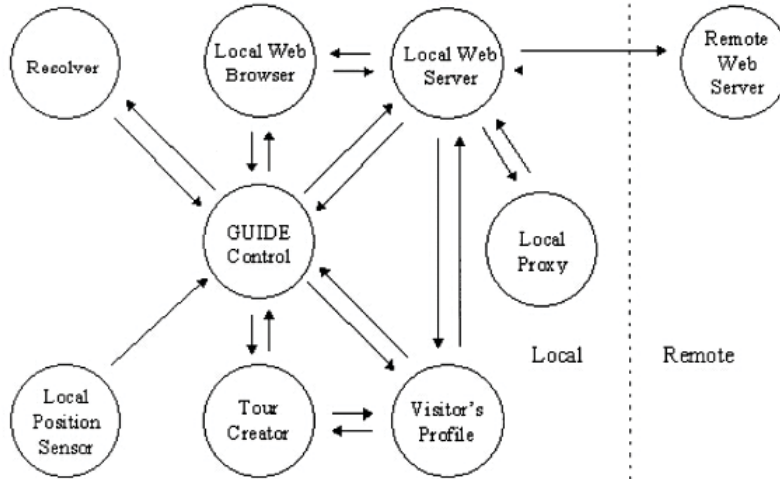


Figure 6: The GUIDE Object Model [CMD99]

spatial and global contexts in computer Go (a 4000 years old game that is very famous in Japan, China and Korea). They revised their object oriented context modeling approach with its inheritance and reusability in order to simplify knowledge representation in complicated domains and systems.

5. Logical Based Models

In logical based models, the context is defined as facts, expressions and rules by using logic theory. Operations for a logic based system, including adding, updating or deleting, depends on logic reasoning or inferencing. All logic based models can be regarded as a high level of formality [SLP04].

In early 1993, McCarthy and his group at Stanford proposed one of the first logic based context modeling approaches namely as *Formalizing Context* [McC93, MB97]. Their objective was to introduce contexts as abstract mathematical entities with properties useful in artificial intelligence. He tried to build a formalization scheme in order to allow simple axioms for common sense phenomena, *e.g.*, axioms for static blocks worlds situations, to be lifted to context involving fewer assumptions, *e.g.*, contexts in which situations change. A significant part the model is the lifting axiom which deduce the truth in one context based on what is true in another context. The basic relation in this approach is $ist(c, p)$, which asserts that the proposition p is true in the context c . For example, the formula $c0 : ist(contextof("SherlockHolmesstories"), "Holmesisadetective")$,

which asserts that it is true in the context of the Sherlock Holmes stories when Holmes is a detective.

Another method that belongs to this category is the multimedia system by Bacon et al. [MB97]. In this system the location can be regarded as one part of the context which are described as facts in a rule based system. The developers use Prolog to implement this system.

6. Ontology Based Models

An ontology is used to formally represent knowledge within a domain. Using ontology for context modeling allows a semantic description of context and share common understanding of the structure of contexts among users, devices, and services [CBJC11]. Ontologies provide a uniform way for describing the models concepts, subconcepts, relations, properties and facts and also propose a way to share the contextual knowledge and reuse the information. They are a powerful tool to model concepts and interrelationships. Besides, using ontology reasoning to evaluate and interpret the contextual information also has an ability to allow computers to determine contextual compatibility and compare contextual facts [KS07].

The CONtext ONtology (CONON) is a representative work of ontology based modeling approaches proposed by Wang et al. [WZGP04]. In the CONON, there are four fundamental context for acquiring information about the executing situation: location, user, activity and computational entity. Those entities are used to show associated information and build the skeleton of context. In order to provide flexible extensibility for adding concepts in application domains, CONON proposes a context model which has two parts (upper ontology and specific ontology). The upper ontology is a high-level which contains general features of context entities. Figure 7 offers a sample of the CONON upper ontology. The context model is constructed by a set of abstract entities, including *Person*, *Activity*, *Computational Entity (CompEntity)* and *Location*, as well as a set of sub-classes. The specific ontology can be seen as a set of ontologies which indicate the details of general concepts and their features in each sub-domain. Figure 8 gives an example of a specific ontology for a smart phone

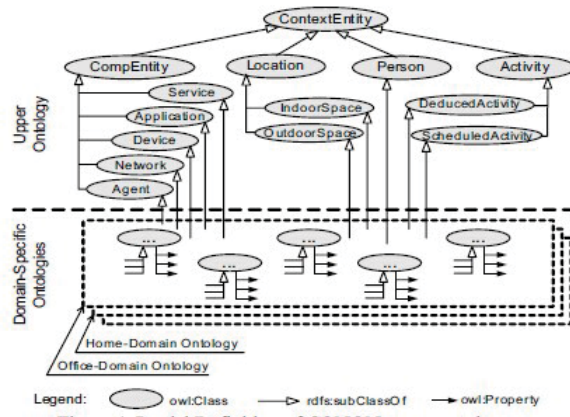


Figure 7: Partial Definition of CONON Upper Ontology [WZGP04]

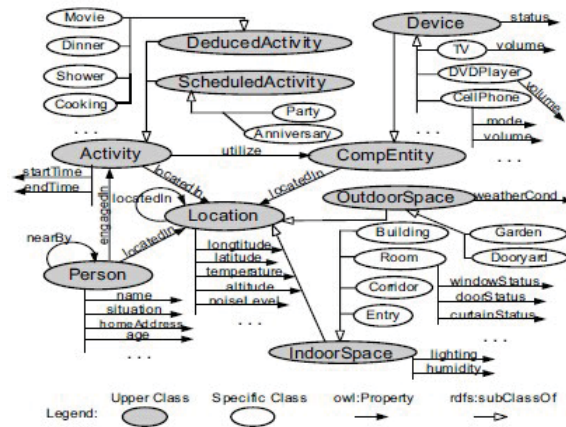


Figure 8: Partial Definition of a Specific Ontology for Home Domain [WZGP04]

home application domain. This specific ontology defines several concrete subclasses for model specific context in a given environment, *e.g.*, the abstract class *OutdoorSpace* of home domain can be classified into two sub-classes *Garden* and *Dooryard*. A context can be processed with logical reasoning mechanism if it is modeled by taking a formal ontology approach. The context reasoning mechanism in CONON is implemented in two categories: ontology reasoning using description logic (DL), and user-defined reasoning using first-order logic. The objective of DL reasoning is to fulfill all the important logical requirements, including concept satisfiability, class subsumption, class consistency and instance checking. On the other hand, user-defined reasoning is more flexible and is used to define reasoning rules which can be adopted to derive user's situation.

There are also other research work on the ontology-based context model. One of the first approaches of modeling ontology-based contexts has been proposed by Öztürk and Aamodt [ÖA97]. They utilize their experience in the field of normalization and formality to examine the difference between *recall* and *recognition*. *Aspect-Scale-ContextInformation (ASC)* [Str03] is another approach belonging to the ontology category. It proposes an uniform way to model the context as concepts, subconcepts and facts. Furthermore, this approach is able to enable contextual knowledge sharing and reuse in an ubiquitous computing system [DB03].

2.3.2 Web Application and Contextual Awareness

Web application

A Web application is any software or application that allows users access through a Web browser [Wik14d]. In other words, a Web application uses a Web browser as a client. The “client” is used in client-server environment to refer to the program the person uses to run the application. A client-server environment is one in which multiple computers share information such as entering information into a database. The “client” is the application used to collect the requests from users, and the “server” is the application used to handle those requests. For a Web application, the “client” is represented by Web browsers. This architecture is named the browser-server model. In this structure, the user interface is achieved through the Web browser, a very small part of the business logic in the front-end (browser), but the main business logic is implemented on the server side (server), the formation of the so-called three-tier structure. This greatly simplifies the load, reducing the cost and effort of system maintenance and upgrade, reducing the overall cost of ownership. All Web applications apply this architecture. The front-end of a Web application is created in browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and depends on Web browsers to display. Its back-end can use various types of programming languages to implement, *e.g.*, Java, C# and PHP. Nowadays, Web applications become more and more popular. The most important reason is their ability to update and maintain Web applications without distributing

and installing other software on user's computers, *i.e.*, the inherent support for cross-platform compatibility [Wik14d]. Ordinary Web applications include online stores, social network website, wiki and many other functions.

Context-aware computing

At present, context-aware computing is a hot topic within HCI (Human Computer Interaction) [Lue00]. It creates a way to use situational and environmental information about people, places and things. This is usually done by using sensor technologies and mobile computing. It is also able to enable computer systems to anticipate users needs and to react in advance [CK⁺00]. Applications encouraged by context-aware computing can discover and take advantage of contextual information. For example, a context-aware shopping assistant system could display the items in the retail store based on the user's location context and also recommend store items depending on the user's profile (Identity Context) [CK⁺00].

Schilit classified Context-aware computing systems into three categories [SAW94]. First, proximate selection is a user-interface technique where the objects located nearby are emphasized or otherwise made easier to choose. Usually, there are three kinds of located-objects for this technique: computer input and output devices that need physical interaction; non-physical objects and services can be accessed from locations; and the set of places users want to find out. Second, automatic contextual reconfiguration is a process of adding new components, removing existing components, or altering the connections between components due to context changes. Typical components are servers and connections are channels to clients. In some cases, people's actions can be often predicted by their situation. In order to explore this fact, contextual information and commands is emerged which can produce different results according to the context in which they are issued. The last category called context-triggered actions which are simple IF-THEN rules used to specify how context-aware systems should adapt.

Web Application and Context Awareness

The front-end of a Web application is built by using the combination of HTML, Javascript and CSS. The latest version of HTML is HTML 5. With the development of HTML, HTML 5 currently supports the Geolocation API directly if the Web browser

implements it. This API defines a standardized way to retrieve the geographical location information for a client-side device [Wik14c]. Therefore, a Web application created by HTML 5 can have location awareness. Location awareness is an important part of the context-awareness mechanism [MJ03]. This refers to the ability of a Web application to determine the context which will be used. A device or an application which has context awareness, such as a smart phone or a Web application, will be able to detect the situation where it is being used. For a context-aware device, *e.g.*, mobile phone, must have sensors to collect the information. However, for an application, *e.g.*, a Web application, it should have a ability to support some APIs to provide the appropriate information like W3C Geolocation API. This API uses IP address, Wi-Fi and Bluetooth MAC address, radio-frequency identification (RFID) as common sources of location information [Wik14c].

2.4 Context-aware Web Service Discovery and Mashup

Web services provide ways to allow users to access software systems through the Internet using standard protocols. In general, there is a Web service provider to publish a service and users can use this service. Service Discovery can be seen as a procedure of finding a proper service for a request. Context-aware Web service discovery is a subset of Service Discovery. It can be defined as the ability to make use of context information to discover the most relevant services for the user. In order to provide context-aware services, context inputs also need to be considered. Thus, the output of context-aware service now also depends on contextual information [SMA07]. The context aware Web service discovery approach is suggested by Wenge Rong and Kecheng Liu [RL10]. They divide context in two categories as explicit and implicit. Explicit context is provided by users directly. The implicit context is collected by the system in automatic or semi-automatic ways, like the location of a user, current time. The context we build in this paper can be regarded as a synthesis of explicit context and implicit context. Context awareness can be divided into four categories depending on the method of collecting context. The categories are personal profile oriented context, usage history oriented context, process oriented context and other context [MC12].

Mashups are more about information sharing and aggregation to make data more useful. By extension, service mashup is one special kind of mashup combining the data of Web services [BDS08]. The data for mashups should have an ability to be accessed permanently by other services. Because of this, mashups are generally client applications or hosted online. There are several kinds of mashups, including business mashups, consumer mashups, and data mashups [Pee09]. Consumer mashups are the most common because their target is the general public. Consumer mashups are used to combine the data from various public sources and reformat it through a simple browser user interface. Data mashups compose similar types of data and information into a single representation. The combination of these resources will build a new Web service which did not previously exist. Business mashups are often applications which combine their own resources, application and data, with other external Web services. They focus data into a single presentation and allow for collaborative action among businesses and developers.

2.5 Service Composition

2.5.1 Definition of Service Composition

Nowadays more and more enterprises are willing to provide their services or applications functionalities on the Internet as Web services. However, only one Web service usually has a limited functionality. In many cases, a single service cannot fulfill the user's request so that several services are needed to be combined by using service composition to achieve a specific goal [CAH06]. For example, if a user wants to travel, it is not sufficient to book a flight, but she should also take care of reserving a hotel, renting a car, being entertained, and so on. Therefore, service composition can be seen as a procedure to combine existing services in order to satisfy the functionality required by the user. The output of the service composition process is also a service called composite service. Composite services are recursively defined as an aggregation of elementary and composite services [KL03]. In other words, from a user's viewpoint, although this composition is comprised of several services, it can still be considered as a simple service.

2.5.2 Approaches for Service Composition

Service composition is a highly complex task which is difficult to deal with manually. There are several reasons for this [RS05]. First, the number of available Web services have increased greatly in recent years. Second, Web services can update and revise without interruptions. Therefore, the composition system needs to have the ability to detect the updating at runtime. Lastly, there does not exist a unified language to define and evaluate Web services because of different sources of Web services. Consequently, it is important to use an automated way or semi-automated way to build composite Web services. Generally, methods for Web service composition can be divided into two categories. Some Web service composition can use workflow techniques, others use planning methods [RS05].

Workflow-based composition techniques have two branches, static workflow and dynamic workflow [RS05]. The static workflow means a requester defines an abstract process before the composition starts. The dynamic workflow means the composition creates the process and selects services automatically. The requester needs to specify constraints, like the dependency of services, the users' preference and so on. EFlow [CIJ⁺00] is a platform that uses a static workflow generation method to do service composition. The Polymorphic Process Model (PPM) [SGCB00] uses a method that combines the static and dynamic service composition. Many research efforts tackling Web service composition problems via AI planning have been reported [DPAM07]. OWL-S (Semantic Markup for Web Services) [MBH⁺04] is a Web service language that indicates the direct connection with planning. PDDL and rule-based planning [MBE03] are also two useful planning methods used to tackle Web service composition. Some service composition problems can be solved using a planning graph [YZ08, ZY08]. The planning graph will expand iteratively by single levels during iteration. The whole process will work until the proposition set contains all the goal propositions. If the planning problem does not have a solution, the planning graph will not generate output. If there is a solution, the planning graph will show a sequence of actions, which can be regarded as the problem's solution.

2.5.3 Planning Problem

Classical planning deals with finding a sequence of actions that transfer the world from some initial state to a desired state [BSR10]. Classical planning problems can be defined by using PDDL and use PDDL planner. PDDL is a standard encoding language for classical planning tasks [MGH⁺98]. Temporal planning can be solved by similar methods for classical planning. The main difference between temporal planning and classical planning is that there is the possibility of several temporally overlapping actions with a concurrent duration, so that the definition of a state has to include information about the current absolute time and how far the execution of each active action has proceeded [Wik14a]. The name probabilistic planning usually means to plan with probabilistic action effects, with a description to optimize the success probability of the plan. In other words, probabilistic planning is able to compute a plan that handles many or even all foreseeable contingencies [LT07]. Probabilistic planning can be solved with iterative methods such as value iteration and policy iteration when the state space is sufficiently small [Wik14a]. Some planning problems can also be solved by applying constraint satisfaction techniques. The most important steps requires the use of a constraint model to encode a planning problem. For instance, some problems can adopt the straightforward constraint model that has been described in [GNT04] and a more advanced model called CSP-PLAN [LB03].

2.6 Information Aggregation

Information Aggregation is a service that collects relevant information from multiple sources to help individuals and businesses use information effectively by analyzing the aggregated information for special purposes using Internet technologies [ZSM01]. The source providers are named aggregators. A Web aggregator is an entity that can transparently collect and analyze information from different Web data sources. Transparency and analysis are two key characteristics for a Web aggregator [MSF⁺00]. Transparency means an aggregator should access the information of the data sources by using an ordinary way like a normal user. Analysis indicates that an aggregator should synthesize value-added information based on post-aggregation analysis instead of only simply presenting the data.

There are a lot of information aggregation applications over the Internet currently,

such as Yelp, Foursquare, YellowPages, Tripadvisor [Tri13] and Expedia [Exp13]. There are two similarities among those applications. First, those applications are service providers used to aggregate business services. Second, a user can use some keywords to get services through those applications. Some applications can also be used to build travel plans like Tripadvisor and Expedia. Those travel plans just include few types of services, *e.g.*, flight, hotel and car. Moreover, those travel plans have long time spans (more than one day).

2.7 A Motivating Example: Personal Entertainment Planner

The objective of the Personal Entertainment Planner is to collect the users' interests or environment properties (user context), discover nearby related services (service discovery), and make a plan in the form of business process for the user (service composition). With the rapid development of the network, using the Internet has become an inseparable part of our daily life. Many people use applications or services through the Internet to help them in their daily activities. One scenario that interested us is as follows. When you travel to a new city for a business trip, you would most likely want to experience some entertainment in the evening after a long day of work. At that time, it is useful to have a Web application to guide you on how to spend your spare time. If you carry a smart device (smart phone or tablet) which is connected to the Internet, you can use the device to run the application easily without installing any additional programs. Suppose you can input the time period (*e.g.*, from 7:00 PM to 11:00 PM this evening), location (*e.g.*, "within 2 km of my location"), budget (*e.g.*, \$100), and the types of activities you want to take (*e.g.*, "movie, restaurant"). With the above information plus other information the Web app can access, such as the current location and time, the Entertainment Planner can discover the services according to those constraints, and is expected to give the following options (with real business names in Montreal):

1. Plan 1: dinner at Restaurant L'Autre Saison from 7:00 - 8:00; Watching movie "The help" at cinema "Cinema Banque Scotia Montreal" from 8:45-11:00;
2. Plan 2: Dinner at Seven Night Club and watch the Hockey game "Canadiens

vs. Boston Bruins” from 7:30 to 11:00;

The user’s contextual information can be collected easily. It is difficult to use the context properly to do service composition and service discovery. In order to use context in the composition algorithm, we need to build a formal model for the context information. Additionally, some operations should be defined over the contexts to let the system states be transferred. The composition algorithm should have the ability to handle services from different domains. Moreover, the composition algorithm should not only generate several solutions (plans) but also guarantee the quality of those solutions (plans), *e.g.*, variety and satisfaction. Additionally, it is necessary to ensure that the algorithm should produce not only solutions that satisfied the user’s requirements perfectly but also “good enough” solutions satisfying the user’s requirements partially so we need to add soft constraints in the algorithm. For service discovery, UDDI [OAS07] is not an option because public UDDI servers are practically unavailable. We need to use normal service engines or service index sites to discover these services.

Chapter 3

Models for Contexts and Services

3.1 Context Model

Dey defines context as “any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.” [Dey01]. For the Web application, context is the information automatically detected by the Web browser, *e.g.*, the location, the identity, and the current time, or provided by the user *e.g.*, the budget and the range of moving that the user is willing to move around. What to include in the context depends on the given application scenario. For example, whether weather should be included in a context depends on what kind of scenarios we are studying. If it is an indoor mobile tour guide, weather does not affect the application, because it is being used indoors. In computer science, the state of a computer system is a technical term for all the stored information, at a given instant in time, to which the system has access [HH12]. In this thesis, a context can be modeled a system state representation. It includes the attributes of entities in the surrounding environment of a user or simply the attributes to describe a system state.

There are several context models, including key-value models, markup schema models, object oriented models, graph models, ontology models, and logic models (see Section 2.3.1). In this section, we propose a model with its own constraints to describe various contexts in a Web application presented in Section 2.7. Our context can be regarded as a key-value model plus predicate logic expressions and an ontology,

which uses low level sensor data (*e.g.*, geolocation data), as well as high level data in predicate logic (*e.g.*, whether a user has seen a movie). Although our model is based on defined ontology, our context is different from an ontology model. Ontology models define low-level context and high-level context using logical reasoning, but there is no reasoning tool embedded in our mechanism. We create an ontology to describe the concepts (types) and the relationship between the concepts in this problem. The concepts (types) are used for defining variables. The set of constraints are used to provide an evaluation of the context. In our work, we expect our context model to have the expressive power to handle soft constraints and hard constraints over real values. At the same time, our context model should be adopted easily by a compositional algorithm. But no existing context models have this capacity. Hence, we propose a model with constraints for the context presentation.

Firstly, we build an ontology for modeling the problem as in Table 2. The ontology gives us the vocabulary of the concepts and the relationship between the concepts when we analyze the problem. A type can be a subtype of another type.

This ontology is used for the entertainment planner, which is represented as a domain terminology for the motivating example presented in Section 2.7. In Table 2, all types are *domain independent*, except subtypes of “service”. The subtypes of “service” are *domain dependent* and can be extended easily. In this section, we only use four service types as a sample: Movie, Restaurant, Shopping and Direction. A direction is one service that indicates the direction between two locations by the appointed travel model like “driving”. People can add new service types according to their domain.

Based on the domain related ontology, we define the variables and the constraints to be employed by the application.

Definition 1 A *variable* is a tuple $\langle varName, dataType, ontologyType \rangle$.

In Definition 1, *varName* is a symbol used to represent a variable. *dataType* is the actual data type for the value of the variable, which is defined below:

$$dataType := date \mid number \mid set \mid string$$

For example, “date” is used to represent the data type of a time point. *ontologyType* is a type or subtype in Table 2 for the semantic meaning of the variable. We can find the relevant values of *ontologyType* in Table 2.

Table 2: Ontology for Context

type	subtype	sub-sub-type
Location	Current Location	
	Destination	
	Start Location	
	Distance	
Money	Cost	
	Budget	
Time	TimePoint	StartTimePoint
		EndTimePoint
	Duration	
Service	Movie	
	Restaurant	
	Shopping	
	Direction	Driving
		Walking
		Bicycling
Record ¹	ActivityTypeRecord	
	ActivityRecord	
Preference		

¹ An *activityTypeRecord* is used to record the types of services, An *ActivityRecord* is used to record those actual activities, see example *activityTypeRecord*={"movie", "restaurant"}.

Example 1 *Budget* is used to indicate the money the user currently has. Then, *variableName* = *budget*, *dataType* = *number*, and *ontologyType* = *budget*.

All variables used in this thesis are listed in Table 3 according to the Definition 1.

Definition 2 A *constraint* is represented as an equality or an inequality of variable mappings. Each constraint has a value to indicate the degree of satisfaction.

There are two kinds of **constraints**, hard constraints and soft constraints. A **hard constraint** defines prohibited regions of variable assignments. They are the constraints that must be satisfied. The value of a hard constraint equals 0 or 1. 0 means this constraint is dissatisfied. A **soft constraint** merely imposes a penalty on certain assignments rather than prohibiting them [Sri13]. Each soft constraint is

assigned a value between 0 and 1. With a soft constraint, the value of the constraint can reflect how well the user is satisfied. 1 means the constraint is completely satisfied, $[0,1]$ means somehow satisfied [SFV⁺95]. 0 means unsatisfied, which is prohibited as in a hard constraint. We introduce soft constraints because they can model daily situations more accurately. For example, the budgeted money and time you plan to spend on a dinner can be modeled as soft constraints, as a person commonly allows themselves to spend a little bit more money and time than the budget. If the budget is well exceeded, the value can be set to 0.

Definition 3 A *context* is a set of variable mappings.

Common attributes of a context X :

$X.location, X.totalMoneyCost, X.time, X.activityTypeExp,$
 $X.activityExp, X.totalTimeCost$

The attributes $X.location, X.totalMoneyCost, X.time, X.totalTimeCost$ are used to represent low level data. $X.activityTypeExp$ and $X.activityExp$ indicate the high level data in predicate logic.

Example 2 For every service S , $X.activityExp$ includes S means S has been taken by the user. This can be illustrated by a predicate logic $\forall S((S \subseteq X.activityExp) \rightarrow User(S))$. $User(S)$ means the user has already taken the service S .

Example 3 Table 4 is a context for a user. We know that the user is at location L , has spent C amount money and D time, and has watched a movie just now. The current time is T .

Based on the definition of **constraint** and **context**, we can use constraints to evaluate one context. Those constraints include hard constraints and soft constraints. Hard constraints are those which we definitely want to be satisfied. In other words, the value of those constraints must be 1. These relate to the successful assembly of a mechanism. The value of each soft constraint indicate its degree of satisfaction. If the value of one soft constraint is 1, this means the constraint be satisfied perfectly. Otherwise, we will calculate the penalty to represent its degree of satisfaction. If the value of one soft constraint is less than 1, there still has a possibility that the target context is acceptable. We can understand that hard constraints express restrictions and soft constraints express preferences.

3.2 Service Model

Definition 4 A **service** a is a tuple $\langle Pre_a, Attr_a \rangle$, where Pre_a is a finite set of preconditions $Attr_a$ is a set of attributes of a .

Common attributes of service a :

$a.startLocation, a.destination, a.startTime, a.endTime,$
 $a.money, a.type, a.duration$

$startLocation$ means the location service a plan to begin, and $destination$ is the location service a plan to finish. $startTime$ is a time point that this service a plan to begin. $endTime$ is a time point that this service a will finish. $money$ means the expense of taking this service. The attribute $type$ represents the kind of one service, e.g., “movie”. $duration$ shows the time expense of taking service a .

This model of service is easy to extend to adapt to different kinds of services, e.g., add new attributes and change preconditions. Pre_a can be seen as a set of constraints. Pre_a includes the preconditions to execute a . For service movie, its precondition is “movie” and it is not being executed (not included in $activityTypeExp$) and we are able to go to the “movie”. Moreover, the user’s location should match the start location of the “movie”. Therefore its preconditions can be expressed as:

$$Pre_a = \langle \{ \text{“movie”} \not\subseteq activityTypeExp \}, \{ location = startLocation \} \rangle \quad (1)$$

If a service a is applicable to a context X we can apply this service to the context.

Definition 5 A service a is **applicable** to X , denoted as $X \succ a$, if the preconditions of a are satisfied by X .

If we apply a service a to a context X , the context will be changed to X' . We can calculate X' as the following.

Definition 6 Assume a service a is applicable to a context X , $X \succ a$. A new context X' is transformed from the context X after applying a is denoted as $X \xrightarrow{a} X'$. The context X' applies the following assignments for variables.

- $X'.location = a.destination;$
- $X'.totalMoneyCost = X.totalMoneyCost + a.money;$

- $X'.time = a.endTime;$
- $X'.totalTimeCost = (X'.time - X.time) + X.totalTimeCost;$
- $X'.activityTypeExp = X.activityTypeExp \cup a.type^1;$
- $X'.activityExp = X.activityExp \cup a;$

3.3 Context Evaluation

We can use some constraints to evaluate a context X . Before we talk about the constraints, we need to build a user query for service composition as a motivation example to create constraints.

Definition 7 A *user query* is a set of requirements that the target context needs to satisfy.

A user query can be modeled as a set of variable mappings, for example, one user query is as below:

$$\begin{aligned}
 userQuery = \langle &targetActivityType = \{“movie”, “restaurant”\}, timeBudget = 120, \\
 &time = 12 : 00, budget = 100, location = “1417 Du Fort, Montreal”, \\
 &targetArea = “H3H2N7”, userPreference = “balanced”, \\
 &maxDistance = “2km”, travelMode = “driving” \rangle
 \end{aligned}
 \tag{2}$$

In *userQuery*, *duration* is the time budget and *budget* represents the budget. *activityType* indicates the types of services the user want to take. *targetArea* shows the center of the area where the user want to entertain. *maxDistance* is the radius. *userPreference* indicates the user’s preference for taking a list of services, *i.e.*, “balanced”, “budget_first”, or “timeBudget_first”. Table 5 defines the relation between preferences and real (time) cost. Each plan should fulfill with the user’s preference. Then, several constraints can be built to evaluate a context based on *userQuery*.

We use the following rules to evaluate one context X .

¹If *a.type* is not equal to “direction”, $X'.activityTypeExp = (X.activityTypeExp \cup a.type) - \{“direction”\}$. Because each normal service (except “direction” service) can only be taken after taking a direction service.

- The distance between one context's location and user's *targetArea* location cannot exceed *maxDistance*. *locationConstraint* is a variable for one context used to indicate the satisfaction of this constraint. *locationConstraint* = 0 means the distance is larger than *maxDistance*. *locationConstraint* = 1 means the distance is smaller than or equal to *maxDistance*.

$$offset_{lc} = |X.location - userQuery.targetArea| \quad (3)$$

$$locationConstraint = \begin{cases} 1 & \text{if } (offset_{lc} \leq userQuery.maxDistance) \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

- *timeConstraint* = *tc*(*TC_x*, *T_g*), where *timeConstraint* represent the degree of the time budget satisfaction from initial context to the context *X*, *TC_x* is the totalTimeCost of *X*, and *T_g* is the time budget, *offset_{tc}* is the difference between *TC_x* and *T_g*. This a soft constraint, we use function *tc* to calculate the penalty and its value. As *timeConstraint* grows larger from 0 to 1, the time usage will be closer to the time budget. *timeConstraint* = 0 means the time usage is 0 or far beyond the time budget. *timeConstraint* = 1 means the time usage meets the time budget perfectly.

$$offset_{tc} = |T_g - TC_x| \quad (5)$$

$$tc = 1 - \frac{offset_{tc}}{T_g} \quad (6)$$

$$tc(TC_x, T_g) = \begin{cases} tc & \text{if } (offset_{tc} \leq T_g) \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

- *costConstraint* = *cc*(*MC_x*, *C_g*), where *costConstraint* represent the degree of the budget satisfaction from initial context to the context *X*, *MC_x* is the totoalMoneyCost of *X*, and *C_g* is the budget, *offset_{cc}* is the difference between *MC_x* and *C_g*. This a soft constraint, we use function *cc* to calculate the penalty and its value. As *costConstraint* grows larger from 0 to 1, the money usage will be closer to the budget. *costConstraint* = 0 means the money usage is 0 or far beyond the budget. *costConstraint* = 1 means the money usage meets the budget perfectly.

$$offset_{cc} = |C_g - MC_x| \quad (8)$$

$$cc = 1 - \frac{offset_{cc}}{C_g} \quad (9)$$

$$cc(MC_x, C_g) = \begin{cases} cc & \text{if } (offset_{cc} \leq C_g) \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

- *completeness* = $comp(actTypes_x, actType_g)$, where *completeness* for context X represents the degree of completion of the activity types that user input. $actTypes_x$ is the *activityTypeExp*² of context X and $actType_g$ is the *targetActivityType* in *userQuery*. *completeness* = 1 means that the activity types in user’s inputs are fulfilled from the initial context to the context X .

$$comp(actTypes_x, actType_g) = \frac{|actTypes_x|}{|actType_g|} \quad (11)$$

Based on Definition 4, “duration” is one of the attributes of the services. A duration can be seen as a time interval with a start time point and an end time point of a service. In this paper, we consider one human user as the plan executor. Therefore, the services in one plan are in sequential order. When planning, we need to consider how the two services can be connected. In other words, we need to evaluate the time connection between the two contexts. Allen’s interval algebra [All83] defines possible relations between time intervals (Figure 9). In these seven relations, we consider the latter four relations are not feasible, because they need a human user to execute two tasks at the same time. We consider “X meets Y” to be an ideal circumstance. “X takes place before Y” is feasible if the gap between the two services is not too long. Practically, the gap should be less than the duration of the second service. “X overlaps with Y” is feasible, if the overlap between the two services is not too long. This is very practical when planning human activities. For example, a user normally needs one hour to have dinner. If a movie overlaps with the dinner time, the user can possibly rush to finish the dinner or delay going to the cinema, so that both activities can be done. Again, we constrain the overlapping or the gap should be less than the duration of the second service.

We use a variable *serviceConnect* sc to describe how two services are connected (connection between time intervals of two services).

² $actTypes_x$ will include the types in *activityTypeExp* except “Direction”, if *activityTypeExp* contains “Direction”, because *targetActivityType* does not include “Direction”.



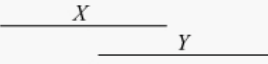
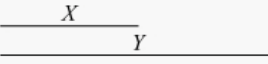
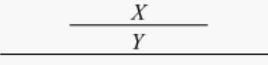
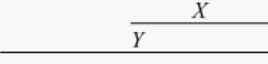
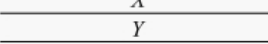
Relation	Illustration	Interpretation
$X < Y$ $Y > X$		X takes place before Y
$X m Y$ $Y m i X$		X meets Y (<i>i</i> stands for <i>inverse</i>)
$X o Y$ $Y o i X$		X overlaps with Y
$X s Y$ $Y s i X$		X starts Y
$X d Y$ $Y d i X$		X during Y
$X f Y$ $Y f i X$		X finishes Y
$X = Y$		X is equal to Y

Figure 9: Possible Relations Between Two Time Intervals [Wik13]

Definition 8 Assume two services a_1 and a_2 has durations $D_x = [s_1, e_1]$ and $D_y = [s_2, e_2]$ ³ respectively. The variable **serviceConnect** sc is calculated using Equation 14, representing the degree of time intervals connection between the two services.

We define $offset_{con} = |e_1 - s_2|$. It is the evaluation of the gap or the overlapping between a_1 and a_2 . The best circumstance is that $offset_{con} = 0$, which is “X meets Y”. We restrict this value to less than the duration of a_2 , *i.e.*, $e_2 - s_2$. Therefore, the feasible circumstance is that $0 \leq offset_{con} \leq e_2 - s_2$, otherwise, it is a infeasible circumstance. We use $s = 1 - offset_{con} / (e_2 - s_2)$ to normalize the value to be between [0 1]. Then, we have the equation 14 to evaluate the feasibility. When it is a feasible circumstance, sc equals to s . The higher the value, the better. When it is an infeasible circumstance, sc equals to 0.

$$offset_{con} = |e_1 - s_2| \quad (12)$$

$$s = 1 - \frac{offset_{con}}{e_2 - s_2} \quad (13)$$

³ s_1 and e_1 in D_x represent the start time point of a_1 and end time point of a_1 separately. s_2 and e_2 in D_y represent the start time point of a_2 and end time point of a_2 separately

$$sc = \begin{cases} s & \text{if } (e_1 < e_2 \wedge \\ & s_1 < s_2 \wedge \\ & offset_{con} < e_2 - s_2) \\ 0 & \text{otherwise} \end{cases} \quad (14)$$

Assume two services a_{i-1}, a_i can generate a process $X_{i-2} \xrightarrow{a_{i-1}} X_{i-1} \xrightarrow{a_i} X_i (i \geq 2)$ from the Definition 6. Based on Definition 8, the degree of time intervals connection between the two services a_{i-1}, a_i can be calculated by using X_{i-1}, X_i . We can use this as a constraint for the context X_i .

Definition 9 The constraint **connectConstraint** for one context X_i represents the degree of time intervals connection between two services a_{i-1}, a_i .

We can calculate the value of *connectConstraint* by using Equation 13 and 14 and sc is the value of *connectConstraint*. There are two special cases for the calculation of *connectConstraint*. Firstly, if the context X_i is an initial context ($i = 0$) which is not transformed from other context after applying one service, we define the value of *connectConstraint* for X_i is 1. Next, if $i = 1$, there exists $X_{i-1} \xrightarrow{a_i} X_i$, we calculate the value of *connectConstraint* for X_i by using Equation 16 and 17.

$$offset_{xcon} = |X_i.time - a_i.startTime| \quad (15)$$

$$s = 1 - \frac{offset_{xcon}}{a_i.duration} \quad (16)$$

$$X_i.connectConstraint = \begin{cases} s & \text{if } (offset_{xcon} < a_i.duration) \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

Definition 10 The *globalConnect* for one context X_i is used to represent the connection degree among the services applied from the initial context to the context X_i .

The value of *globalConnect* in the initial context X_0 is 1. For other circumstances, if one context X_i is not an initial context, it can be generated by using the procedure $X_{i-1} \xrightarrow{a_i} X_i (i \geq 1)$ from Definition 6. We use Equation 18 to calculate the value of *globalConnect* for X_i .

$$X_i.globalConnect = \begin{cases} X_{i-1}.globalConnect & \text{if } (X_i.connectConstraint \\ & \geq X_{i-1}.globalConnect) \\ X_i.connectConstraint & \text{if } (X_i.connectConstraint \\ & < X_{i-1}.globalConnect) \end{cases} \quad (18)$$

Definition 11 A C is a set of constraints to evaluate one context including *locationConstraint*, *completeness*, *globalConnect*, *costConstraint*, *timeConstraint*.

Soft constraints are assigned with a numeric value from 0 to 1. Therefore, we can define a variable ω , which contains the value of constraints in C , to represent the quality of one context.

Definition 12 A ω for one context X is a tuple $\langle lc, comp, gc, mc, tc \rangle$, where

- lc is the value of X 's constraint *locationConstraint*;
- $comp$ is the value of X 's constraint *completeness*;
- gc is the value of X 's constraint *globalConnect*;
- mc is the value of X 's soft constraint *costConstraint* ;
- tc is the value of X 's soft constraint *timeConstraint* ;

Definition 13 The **value** of ω for one context ω_v can be calculated by using equation 19. Two contexts' ω e.g., ω_1, ω_2 can be compared by using their values.

$$\omega_v = (comp * |userQuery.targetActivityType| + (gc + mc + tc)/3) * lc \quad (19)$$

3.4 Comparison with Existing Context Models

Comparing to our context model, the existing context modelling approaches, including key-value models, markup schema models, graphical models, object oriented models, logical based models and ontology based models are not suitable for our context model.

- Key-value models use key-value pairs to model the context. The key-value pairs are used to define the properties and their values. Context-aware applications can only use key-value models to implement simple reasoning, *e.g.*, IF-THEN rules. In addition, although key-value pairs are easy to manage, they do not have an ability to handle complex context information. The context information in our work is more complicated than key-value pairs, *i.e.*, domain information and predicate logic. Therefore, we reuse the key-value model as a part of our context model to represent low level data.
- Markup schema models define the context into hierarchical data structure including of markup tags with attributes. The attribute of the markup tags is recursively defined by other markup tags. Markup schema models do not have an ability to express predicate logic. However, our context model uses low level sensor data (*e.g.*, geolocation data), as well as high level data in predicate logic (*e.g.*, whether a user has seen a movie). Thus, our context model does not make use of Markup schema models.
- The objective of Object oriented models is taking the benefit of the encapsulation and reusability of the object oriented approach. Only the specified interfaces are defined to have access to the context information. In our context model, it is unnecessary to hide the details of context processing. Hence, our context model does not make use of object oriented models.
- Graphical models can be seen as extensions of object oriented models. Graphical models use graphs with graphical notations to represent context information and the dependency relations between classes or entities. Graph models are mainly used to describe the structure of contextual knowledge and derive some code or an ER-model from the model, which is valuable in the sense of the applicability requirement. Whereas, our context model concentrates more on data. Therefore, our context model does not make use of Graph models.
- Ontology models construct a context ontology using the Web Ontology Language (OWL). In ontology models, context reasoning is implemented by Description Logics (DL) reasoning to fulfil logical requirements. But there is no reasoning tool embedded on our mechanism. We just create a domain independent ontology

to describe the concepts (types) and the relationship between the concepts. As a result, our context model only make use of an ontology not a complete ontology model.

- The objective of Logic based models is to propose an adequate theory of reasoning with contexts. However, there is no reasoning tool for our mechanism and only a part of data are predicate logics. Therefore, we reuse the way to express logics in this kind of model rather than fully adopt this model.

Apart from the existing context modeling approaches, we use low level sensor data as well as high level data in predicate logic to describe our context and provide constraints to evaluate our context. The features of our context model are as follows:

- No reasoning tools

One characteristic of context computing is that an operation can be triggered by the current context. In our context model, the operations to be executed are based on several predefined rules. Also, there are no reasoning tools embedded on our mechanism.

- Operational model

Our context model can be seen as an operational model. Arithmetic operations to be easily used by a composition algorithm, based different principles, such as search, planning, or integer programming.

- Mixed data type

Our context is able handle logic values as well as real numbers.

- Extensible and reusable for different scenarios

Though our research is limited to the scope of service composition, the service composition scenarios can be different, e.g. travel, office workflow etc. Therefore, we present ontology for generic service attributes, and this ontology can be used for extension to different scenarios.

Table 3: Variables

variableName	dataType	ontologyType
location	string	Current Location
targetArea ¹	string	Location
startLocation	string	Start Location
destination	string	Destination
maxDistance	number	Distance
budget	number	Budget
time	date	TimePoint
duration	number	Duration
travelMode	string	Direction
targetActivityType ²	set	ActivityTypeRecord
distance	number	Distance
money	number	Cost
startTime	date	StartTimePoint
endTime	date	EndTimePoint
totalTimeCost ³	number	Duration
totoalMoneyCost ⁴	number	Cost
activityTypeExp	set	ActivityTypeRecord
activityExp	set	ActivityRecord
userPreference	string	Preference

¹ *targetArea* shows the center of the area where the user want to do entertainment.

² *targetActivityType* indicates the types of serices the user want to take.

³ *totalTimeCost* represents the user’s total time cost from the start time to the present time.

⁴ *totalMoneyCost* represents the user’s total money cost from the start time to the present time.

Table 4: A Context Example

Variable Name	Data Type	Ontology Type	Sample Value
location	string	Location	L
time	real	TimePoint	T
totoalMoneyCost	number	Cost	C
totalTimeCost	real	Duration	D
activityTypeExp	set	ActivityTypeRecord	{“movie”}

Table 5: Relation Between Preferences and Real (time) Cost

Preference	Cost	Time cost
balanced	$(1 \pm 15\%) * budget$	$(1 \pm 15\%) * timeBudget$
budget_first	$(1 \pm 5\%) * budget$	$(1 \pm 15\%) * timeBudget$
timeBudget_first	$(1 \pm 15\%) * budget$	$(1 \pm 5\%) * timeBudget$

Chapter 4

Service Discovery and Mashup

Through the user interface of Web applications, the user can input business goals. We can build a user query by integrating the business goals, which can also be converted to constraints. We are able to discover related services depending on the user query. Those services will be passed to the service composition to reach the business goals. What we want to do for service discovery and mashup is below:

- We discover the non-electronic services through querying online resources. Any support software available online can be considered a resource. Most non-electronic services providers are common search engines and RESTful data services, hence, our work will focus on those resources rather than UDDI.
- In order to demonstrate the wide coverage of services, we use various online resources (services providers). We use Google Place, Yelp, Foursquare and the Yellow Pages to search for business services, *e.g.*, restaurant and shopping mall. Each type of business services can be found over those four resources. Google Show Time is regarded as an HTML engine that returns movie services. Transportation services can be provided by employing Google Maps which is used to find directions from the original place to the destination.
- We translate the user query into query strings and query criteria.
- We create a database to store services after each search request. The database can be seen as a cache of services. Each time one user does a search, this request will go to the database and make a query first. If the result services we

collect are not enough for doing service composition, we should discover services through querying online resources at that time.

- We build a mashup engine to accept the user inputs and prepare the candidate related services for service composition. The mashup is able to integrate different type of services and formalize the services according to the service model in the previous section. After the mashup procedure, services can be used in the service composition algorithm.

4.1 Service Discovery in Practice

Universal Description, Discovery and Integration (UDDI) is a XML based registry that businesses worldwide can use to list themselves on the Internet, and a mechanism to register and locate Web service applications [OAS07]. Though UDDI server is designed for service discovery, practically no public UDDI server is available. Therefore, we use general purpose search engines such as Google, Yelp or Foursquare to discover services.

There are several Web search engines that collect information about SOAP and RESTful services, *e.g.*, the URL, query format and query examples. Descriptions of several search engines are as below.

- Woogle [DHM⁺04] uses a clustering technique to search the desired Web services that satisfy requirements described as keywords. Woogle retrieves Web services through Web service descriptions registered on UDDI.
- WSExpress [ZZL10] collects Web service description through crawling the Internet. They use both functional and non-functional characteristics to sort the search results.
- Seekda [See12] employs the general search engine Alexa to discover Web services. The services users get are ranked according to not only the similarities to the users' requirements but also qualities of services, *e.g.*, service response time and service reliability.

In this thesis, the services we want to use are real world businesses (non-electronic services), *e.g.*, restaurants, movies, and shopping, rather than electronic services in the

SOA domain. Because of this, we do not choose any existing Web service search engines to discover services. In order to achieve our business goals, we use general purpose search engines to discover services and businesses. Those services discovered by search engines will be adopted in the service composition part. Moreover, we propose a mashup phase to formalize the inputs and the results of search engine services to ensure that the different types of services work together in service composition.

4.2 Available Search Engine Services

Our service discovery application is used for discovering related services from general search engines, which provide non-electronic services. Currently, most of those general search engines only support RESTful services and HTML services, *e.g.*, Google, Yellow pages and Yelp. Therefore, we concentrate on retrieving non-electronic services from RESTful services and HTML services in this thesis.

In order to demonstrate the wide coverage of services, we use various online resources. We use Google Place, Yelp, Foursquare and Yellow Pages to search for business services, *e.g.*, restaurant and shopping mall. Google Show Time as an HTML engine returns movie services. After retrieving services, this application will store those services in the database.

It is possible to parse services in a uniform way (*e.g.*, using JSON or XML) because their request and response formats are similar over the HTTP protocol. Thus, we build a mashup engine to enable different types of services to work together. The inputs of the service mashup engine is a user's query and the outputs of the engine are a list of services that can be employed in the service composition engine.

4.2.1 RESTful Services

Representational State Transfer (REST) has gained widespread attention across the Web as a simpler alternative to SOAP and Web Services Description Language (WSDL)-based Web services. The RESTful services can be seen as a part of SOAP. They focus on resources, which are abstract entities identified by URIs. This basic REST design principle builds a one-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods (POST, GET, PUT, DELETE). The state of one resource is maintained at the client application. In a RESTful service,

the server is responsible for generating responses and for providing an interface that enables the client to maintain a resource state. A resource representation typically reflects the current state of a resource and its attributes, which can be serialized by using specific media types (*e.g.*, XML, HTML, JSON, text, etc).

One pair of sample request and response related to a RESTful service is shown in Table 6. The request is to get the information for a customer. The URI of this request is `example.com/info/customers/1`. From the request, we know that the response should be in XML format. The request is an HTTP GET request. We can see the customer information in XML format from the response.

Table 6: The Request and Response of a RESTful Service

Request:

```
GET /resources/customers/1/HTTP/1.1
Host: example.com
Accept: application/xml
```

Response:

```
HTTP/1.1 200 OK
Date: Sun, 07 Dec 2014 22:29:26 GMT
Server:Apache-Coyote/1.1
Content-Type:application/xml
<?xml version="1.0"?>
<CUSTOMER xmlns:xlink="http://www.w3.org/1999/xlink">
<ID>3 </ID >
. . .
</CUSTOMER >
```

Developers need to write their own code to generate the input request and translate the output messages for invoking one RESTful service, because RESTful services do not have a unified standard to describe their input and output. In order to help developers, each RESTful service should provide development documents as instructions. All kinds of RESTful services we use are shown as below.

Google Places API

Google Places API [Goo13b] is a RESTful service which allows you to query for place information on a variety of categories, such as: establishments, prominent points of interest, geographic locations, and more. You can search for places either by proximity or a text string. The output of the service is in either JSON or XML for parsing by the application. A search request is an HTTP URL of the following format:

```
https://maps.googleapis.com/maps/api/place/service/output?parameters
```

In the above HTTP request, service indicates the particular type of search requests, e.g., Nearby Search Request, Text Search Request and Radar Search Request, and output indicates the response format, which can be JSON or XML. Certain parameters are required to initiate a Search request. Some samples of parameters are listed in Table 7. As is standard in URLs, all parameters are separated using the ampersand (&) character.

Table 7: Google Place API Parameters

Parameter	Meaning
location	latitude/longitude textual value for the place around
radius	distance (in meters) within which to return place results
keyword	a term to be matched against all content
types	restricts the results to places matching at least one of the specified types
...	

Example 4 A request is below, showing a search for Places of type 'food' within a 500m radius of a point in Sydney, Australia, containing the word "cruise" in their name: `https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=-33.86705,151.19573&radius=500&types=food&name=cruise&key=YourOwnKey`. A Place Search returns a list of Places along with summary information about each Place, such as name, address and phone number. The structure of the JSON is listed as below:

```
{
  "geometry" : {
```



```

        "location" : {
            "lat" : -33.867217,
            "lng" : 151.195939
        }
    },
    "name" : "Biaggio Cafe - Pyrmont",
    "rating" : 3.4,
    "types" : [ "cafe", "bar", "restaurant", "food", "establishment" ],
}

```

Yelp v2.0 API

Yelp v2.0 API [Yel13] is a RESTful service which enables access to more relevant search results that more closely match the results on Yelp. It uses a standard and secure authorization protocol (OAuth 1.0a, xAuth). The output of the service is only in JSON format. A search request is an HTTP URL of the following format:

```
http://api.yelp.com/v2/search?parameters
```

Some examples of parameters are listed in Table 8 and all parameters are also separated using the ampersand (&) character.

Table 8: Yelp API Parameters

Parameter	Meaning
term	search term (e.g. “food”, “restaurants”). If term isnt included we search everything
limit	number of business results to return
category_filter	category to filter search results with
radius_filter	search radius in meters
...	

A sample request is below, showing a search for places of term ‘food’ within a 500m radius of a point in the position(37.77493,-122.419415):

```
http://api.yelp.com/v2/search?term=food&radius_filter=500&ll=37.77493,
-122.419415
```

Foursquare Search Venues API

Foursquare Search Venues API [Fou13] is a RESTful service which returns a list of venues near the current location, optionally matching a search term on Foursquare. The user will need a client ID and client secret to make a userless venue search or explore request. A search request is an HTTP URL of the following format:

```
https://api.foursquare.com/v2/venues/search?parameters
```

All of the parameters in Foursquare API are optional. The result will be present in JSON format. Table 9 shows the samples of parameters.

Table 9: Foursquare Search Venues API Parameters

Parameter	Meaning
ll	Latitude and longitude of the user's location
query	a search term to be applied against venue names
limit	number of results to return, up to 50
radius	limit results to venues within this many meters of the specified location
categoryId	a comma separated list of categories to limit results to
...	

For example, one request to search for places of term 'food' within a 500m radius of a point in the position (37.77493,-122.419415) is as below:

```
https://api.foursquare.com/v2/venues/search?query=food&radius=500&ll=37.77493,-122.419415
```

YellowAPI's Places API

YellowAPI's Places API [API13] is a RESTful service which allows you to stream top Canadian local search content into Web applications. All methods in YellowAPI use HTTP GET requests. The responses of YellowAPI can be in JSON or XML. A search request is an HTTP URL of the following format:

```
http://api.yellowapi.com/FindBusiness/?parameters
```

Parameters in YellowAPI can be divided into two categories: required parameters and optional parameters. All parameters are also separated using the ampersand (&) character. In Table 10, we list all the parameters which are required.

Table 10: YellowAPI’s Places API Parameters

Parameter	Meaning
what	a search term which can be a keyword
where	the location to search
fmt	the format of the output (JSON or XML)
apikey	API key for the Places API

A sample request is below, showing a search for places of term “food” in Montreal and the response is in JSON format:

```
http://api.yellowapi.com/FindBusiness/?what=food&where=Montreal&
fmt=JSON&pgLen=1&apikey=samplekey
```

Google Maps Web Services

Google Maps Web Services [Goo13a] are a set of RESTful services which use HTTP requests to specify URLs and passing URL parameters as arguments to the services. The output of the service is in either JSON or XML. A service request is of the following format:

```
https://maps.googleapis.com/maps/api/service/output?parameters
```

In the above HTTP request, service indicates a specific service, *e.g.*, directions or distance matrix, and output shows the response format, which normally is JSON or XML. Some examples of parameters are listed in Table 11.

Table 11: Google Maps Parameters

Parameter	Meaning
origin	address or textual latitude/longitude value
destination	a address or textual latitude/longitude value
mode	mode of transport to use, <i>e.g.</i> , driving
avoid	avoid the indicated features, <i>e.g.</i> , tolls
...	

Example 5 A query for getting XML output is `https://maps.googleapis.com/maps/api/directions/xml?origin=Chicago,IL&destination=Los+Angeles,CA&mode=driving`. The query above is for searching driving directions between two places (from Chicago, IL to Los Angeles, CA). The Directions API can return multi-part directions using a series of waypoints, durations and distance. The structure of the XML is listed as below:

```
<DirectionsResponse>
  <status>OK</status>
  <route>
    <leg>
      <step> . . . </step>
      ...
      ... additional steps of this leg
    </leg>
    ...
    ... additional legs of this route
    <duration> ... </duration>
    <distance> ... </distance>
    <start_location> ... </start_location>
    <end_location> ... </end_location>
    <start_address>Chicago, IL, USA</start_address>
    <end_address>Los Angeles, CA, USA</end_address>
  </route>
</DirectionsResponse>
```

4.2.2 HTML Services and Google Show Time

HTML Services

Currently, there is not a uniform definition for HTML services. We usually consider the Web applications which can return HTML pages as HTML services. Based on this definition, every Web site can be regarded as an HTML service. Because of the wide use of HTML services, the key point for us is to know how to retrieve the information from them in order to accomplish business goals in service composition. The HTML

services adopts the HTTP query (URL + query string) as an input and returns HTML pages as an output. Table 12 describes the detail of request and response for a HTML service. Most of the HTML pages generated from HTML services have a stable structure. Therefore, we are able to program a parser to dispose the response and extract useful information from them.

Table 12: The Request and Response of a HTML Service

Request:

```
GET /resources/customers/1/HTTP/1.1
Host: example.com
Accept: text/html,application/xhtml+xml
```

Response:

```
HTTP/1.1 200 OK
Date: Wed, 10 Dec 2014 00:28:03 GMT
Server:Apache-Coyote/1.1
Content-Type:text/html,application/xhtml+xml
<html>
<head>... </head >
<body>... </body >
</html >
```

Google Show Time

Google Show Time is a part of Google search engine. You can send an HTTP query like <http://www.google.com/movies?near=45.496330,-73.578829> to get a movie schedule near the location you put in the query string. The returned response is in HTML format. We use Jsoup to locate the cinema address and shot times for movies from the HTML response. The returned response in HTML format is shown below:

```
<div id="movie_results">
  <div class="theater">
    <h2 class="name">
      Cinema du Parc
```

```

</h2>
<div class="info">
  3575 avenue du Parc, Montreal, QC, Canada - (514) 281-1900
</div>
<div class="showtimes">
  <div class="movie">
    <div class="name">
      Birdman
    </div>
    <span class="info">
      1 hour 59 minutes - English - IMDb
    </span>
    <div class="times">
      1:30 4:00 6:30 9:00
    </div>
    . . .
  </div></div></div>
</div>

```

4.3 Service Discovery and Mashup

The service discovery and mashup part in our system can be regarded as an information aggregation component. We use this part to discover and integrate services. Then those services will be passed to the service composition engine. Figure 10 is the class diagram of the service discovery mashup to describe the process. The mashup contains three parsers based on their functionalities, *i.e.*, *HttpParser*, *DirectionParser*, *GenericServiceParser*. Each parser implementing a interface called *ServiceParser* searches its corresponding service and convert the particular result of service into a list of general activities, *i.e.*, *MovieService*, *DirectionService* or *GenericService* in a uniform format, which inherits the super class *Activity*. The *HttpParser* invokes the *GoogleShowTimeEngine* through the *MovieProvider* to retrieve the html pages related with movie information and converts those pages into activities. The *DirectionParser* uses *GoogleMapsEngine* through the *DirectionProvider* to generate direction

information and convert it into general activities. The *GenericServiceParser* is used to collect general business services (e.g., restaurants, shopping malls and museums) by invoking those four search engine services through the *GeneralServiceProvider*, i.e., *YelpAPI*, *GooglePlaceAPI*, *YellowpageAPI*, and *FoursquareAPI*. After that, the *GenericServiceParser* will integrate results and convert them into *GenericServices*. *GenericService* can represent many types of business services which have the same properties except the property “type”. The generated activities are part of the inputs of the service composition part.

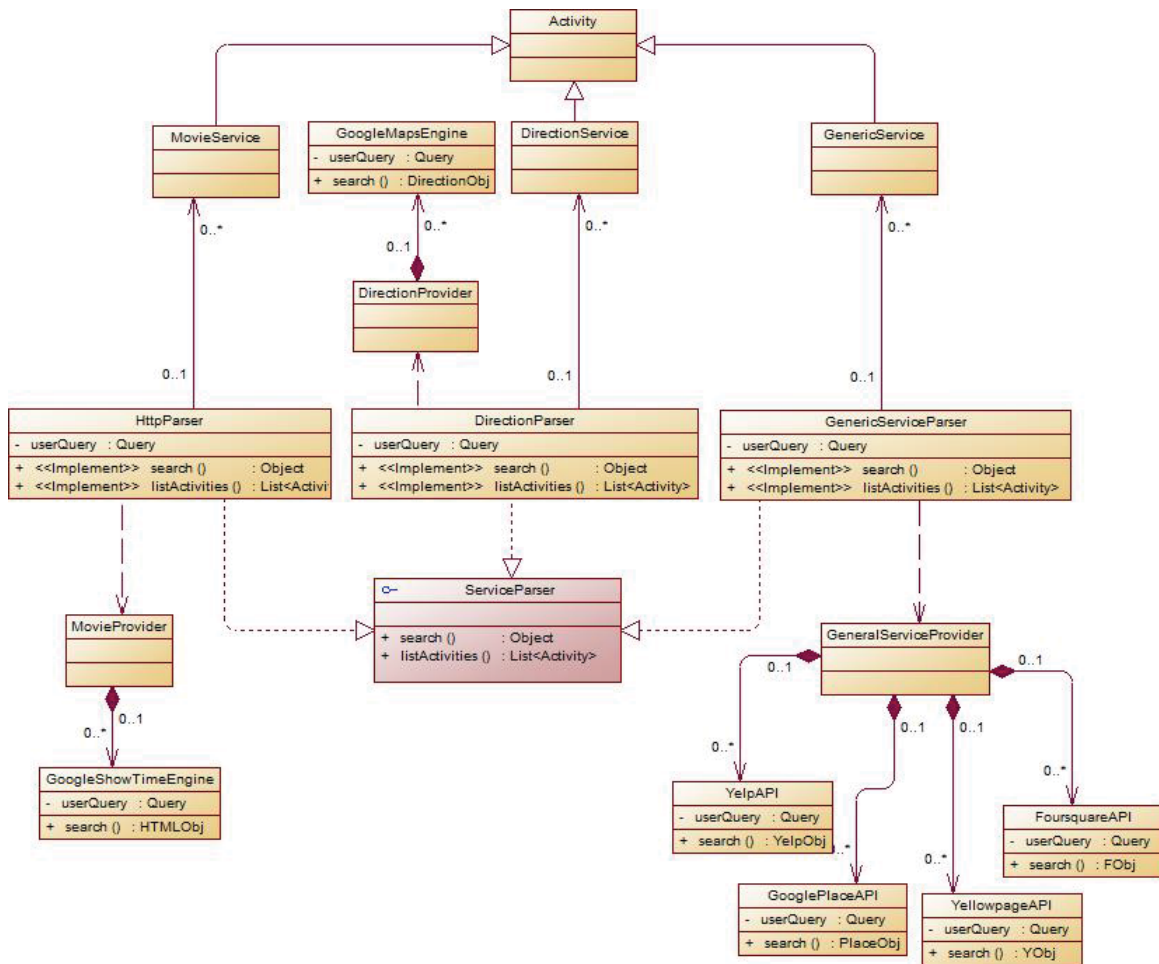


Figure 10: Diagram of Mashup

Figure 11 describes the control flow of the service discovery and mashup component without the database. The service discovery application is used for discovering related services, i.e., generic business services and movie services. This application uses *GeneralServiceProvider* to collect generic business services and parse the information

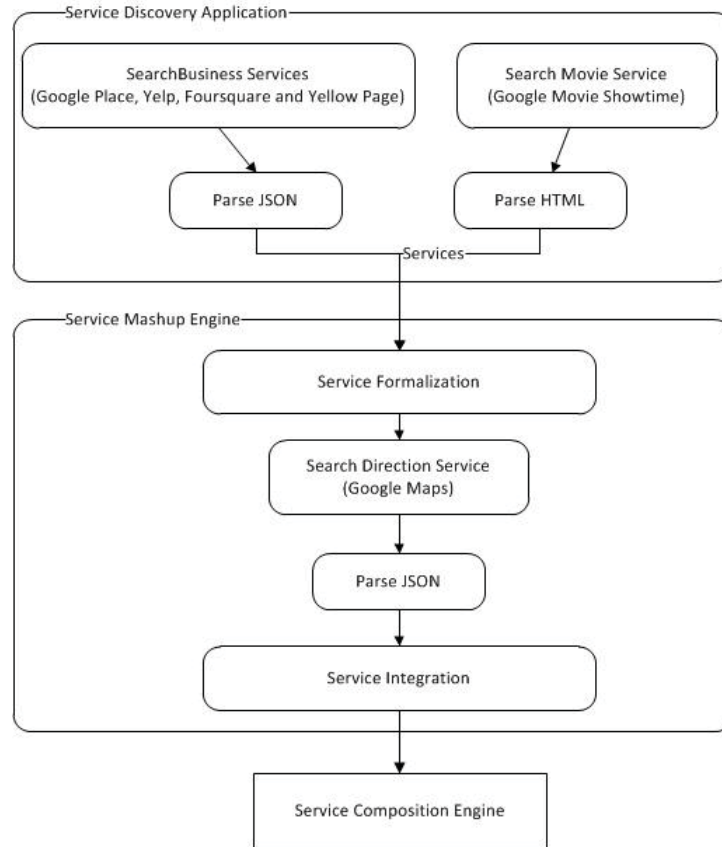


Figure 11: Service Discovery and Mashup

of those services, and also uses *MovieProvider* to retrieve movie services. After retrieving services, this application will store those services in the database. Then, those services will be passed to the mashup engine. The mashup engine accepts the user inputs and prepares the candidate related services for service composition. At first, this engine queries the services from the database using user inputs. If the result services we collect are not enough, the engine will send a request to the service discovery application to retrieve services. After that, the mashup extracts the service information from the query results and formalizes the services according to the service model in the previous section. Next, we use Google Maps Web service by invoking the *DirectionProvider* in service discovery application to find directions from the original place to the destination and make those directions as formalized services. Then all of those formalized services and the user inputs are transferred to the service composition engine to calculate a plan. The service collecting process is implemented by multi-threads depending on the number of service types. Additionally, for the

service formalization, we extract some common service attributes conveniently, which are duration, address, rating, cost, start time and end time. When some values are unavailable, we have predefined the rules to set the value. For example, the cost for a restaurant is \$20 and the cost for a movie is \$10.

Chapter 5

Service Composition

Service composition is used to conduct a procedure to combine existing services in order to satisfy business goals defined by the user that can not be fulfilled by individual services. In our work, all the services in different types for service composition should be uniformed into the same format after service discovery and mashup. Therefore, service composition is able to handle and integrate various kinds of services from different resources with the outputs in a unified format. The target of our service composition is to build a personal entertainment planner that can generate several good entertainment plans according to the user's requirements.

Service composition problems can be solved by adopting planning techniques. During the planning phase, we construct a search tree to describe the problem space. Every state node in the search tree is related to a context. Since our context model can be evaluated by soft constraints, we also use soft constraints to assess state nodes in the search tree. Thus, our search tree is able to handle soft constraints. In the following, we first define the symbols employed in our service composition algorithm. After that, we build a theoretical framework of the planning problem for the service composition based on those symbols. At last, one strategy for doing re-planning is developed, which can do service composition according to a user's context if the user is not satisfied with the original solution.

5.1 Problem Description

Definition 14 A *state* s refers to the present circumstance of a system, which is modeled by one context, i.e., a set of variable mappings, including user’s inputs variables and circumstance variables.

For example, the initial state s_0 for the motivation example in Section 2.7 can be described as:

$$s_0 = \langle activityTypeExp = \{\}, totalTimeCost = 0, totalMoneyCost = 0, location = "H3H2N7", time = 19 : 00 \rangle \quad (20)$$

In the initial context as shown in Equation 20, the user does not take any entertainment services at 19:00. He stays at a place which can be identified by the postcode “H3H2N7”. Both of the money cost and time cost are currently 0.

Based on the definition of state, we could treat one state as one context. Therefore, we are able to use ω to represent the quality of one state.

Definition 15 Suppose there are two states s_1, s_2 . ω_{v1} is the ω value for s_1 and ω_{v2} is the ω value for s_2 . If ω_1 is larger than ω_2 , we can say s_1 is **better than** s_2 .

Definition 16 A *service composition query* is a tuple $\langle s_0, \omega_g, C \rangle$, where

- s_0 is an initial state ;
- ω_g is a ω defined according to one user’s query;
- C is a set of constraints used for evaluation;

In a service composition query, ω_g is used to compare with states’ ω . If the value of a state’s ω is greater than or equal to the value of ω_g , that state will be one goal state. Table 13 presents the different definitions of ω_g based on the preferences (see Table 5) in user’s query. C is the constraints used for assess any state.

Definition 17 The *state transition function* γ of one service $a = \langle Pre_a, Attr_a \rangle$ for any state s is $\gamma(s, a) = s'$, if a is applicable to s , i.e., $s \succ a$.

According to Definition 14 and 17, the *state transition function* $\gamma(s, a) = s'$ can be executed by employing the procedure built in Definition 6 at Section 3.2.

Based on the definitions above, we define the problem of service composition.

Table 13: Definitions of ω_g Based on Preferences

Preference	ω_g
balanced	$\langle lc = 1, comp = 1, gc = 0.95^1, mc = 0.85, tc = 0.85 \rangle$
budget_first	$\langle lc = 1, comp = 1, gc = 0.95, mc = 0.95, tc = 0.85 \rangle$
timeBudget_first	$\langle lc = 1, comp = 1, gc = 0.95, mc = 0.85, tc = 0.95 \rangle$

¹ It is very difficult to make the connection degree among services as 1 (the perfect situation), so we offer 5 percent off to gc.

Definition 18 A *service composition problem* is a tuple $\langle s_0, \gamma, \omega_g, A, C \rangle$, where

- s_0 is an initial state ;
- γ is a state transition function;
- ω_g is a ω extracted from one user's query;
- A is a set of available services;
- C is a set of constraints used for evaluation;

Definition 19 A *solution* π to the service composition problem $\langle s_0, \gamma, \omega_g, A, C \rangle$ is a sequence of services $\langle a_1; \dots; a_n \rangle$, in which each a_i ($i \in [1, n]$) is a service. a_1 is applicable to s_0 . a_i is applicable to $\gamma(s_{i-2}, a_{i-1})$ when $i = [2, n]$. s_t hold at a state $s_t = \gamma(\dots(\gamma(\gamma(s_0, a_1), a_2) \dots a_n))$. From state s_t , a solution $\langle a_1; \dots; a_n \rangle$ can be retrieved.

For instance, the *Plan 1* for the motivating example in Section 2.7 can be illustrated as $\pi = \{a_1, a_2\}$, where a_1 represents the service of having dinner at Restaurant L'Autre Saison and a_2 indicates watching the movie "The Help" at the theater "Cinema Banque Scotia Montreal".

Our goal is to find k good enough solutions for the problem. Based on the Definition 19 and Definition 14, a solution can be retrieved from a state so we give the definition of a good enough solution in this paper.

Definition 20 A solution is a *good enough solution* only if it is retrieved from a state whose ω is better than or equal to ω_g .

A service composition problem is to used produce a business procedure that can generate k states whose ω is better than or equal to ω_g from the initial state s_0 . All the constraints in C are used to evaluate states during the composition process.

5.2 Composition Algorithm

Now we are looking for an algorithm to solve the problem. For this composition problem, the model we use is a planning model. Constraints are used to evaluate the planning path (optimize and restrict solutions). This service composition problem can be seen as a planning problem with constraints [NFF⁺05]. And as we discussed above, most of constraints in this problem are soft constraints. Backtracking Search Algorithm (BSA) is an efficient way to solve that kind of problem [HTD90] [DF99]. In this paper, we adopt the backtracking search algorithm as our planning algorithm. The principal idea of BSA is to construct solutions one component at a time and evaluate such partially constructed candidates as follows. For our problem, we add constraints used to evaluate. If a partially constructed solution can be developed further without violating constraints, it is done by adopting the first remaining applicable option for the next component. If there is no applicable option for the next component, the algorithm backtracks to replace the last component of the partially constructed solution with its next option. If the constructed solution is the target solution, the algorithm can be terminated (if just one solution is required) [LMB07]. This point is significant for our problem because the process needs to be stopped when enough solutions are found.

Based on the definition of this composition problem, our objective is to find k states whose ω should be better than or equal to ω_g . We construct a tree of choices being made, called the *state-space* tree [LMB07], to implement the backtracking search process. The tree for a backtracking algorithm is constructed in the manner of depth-first search. Each node of the tree can be regarded as a state. We build a priority queue Q to store the states found so far whose ω is better than or equal to ω_g . The Q is empty at the beginning. Starting at the root node (initial state), the backtracking search algorithm proceeds by applying services, and comparing their ω value with the ω_g value. If one node's ω is greater than ω_g , which means this node is a goal state, the current state will be pushed into Q and the subtree below this

node has no need to search further. When a node's ω does not fulfill a part of the requirements from ω_g the subtree below this node will also be pruned because a hard constraint is violated and the goal is not going to be achieved on this branch. Then, the algorithm backtracks to its parent node at a higher level in the tree and selects a recently generated node to examine. Otherwise, the algorithm tries to find a better state by generating its child nodes. Those child nodes will be searched in a sequential order. The algorithm terminates when the size of the Q is k , or the whole tree has been explored.

The algorithm has four parameters: s , the current node; Q , a priority queue to store states; seq , seq represents the current level of the search tree and seq is an integer; A , a set of available services.

Algorithm 1 presents the detailed steps of the backtracking search algorithm. The backtracking search algorithm starts from the root node s_0 (initial state). Line 1 defines a flag variable $status$. In lines 2, we check whether the size of Q is equal to k (the number of ideal plans the planner returns). If it is equal, the process is complete ($status \leftarrow PROCESS_FINISH$). If the process is not terminated ($status = PROCESS_CONTINUE$), we will compare gc in the ω of s with ω_g (lines 7 to 9). If gc in the ω of s is larger than the gc in ω_g ($isPrune \leftarrow false$), this state can be recorded as a candidate. Next, if lc in the ω of s is equal to 0², we prune this state node and its subtree (line 10 to 12). The two steps above can be seen as pre-checks for the current state in order to improve the efficiency of this algorithm. After that, if $isPrune \leftarrow false$, we compare the ω of the current state with the ω_g and use Algorithm 2 to check the similarities between the current state and previous states in Q , because we prefer to give the users different plans rather than several very similar ones. If the ω of the current state is greater than or equal to ω_g and the current state is not similar with any one state in Q , the current state will be pushed it into Q . The subtree below this node is pruned ($status \leftarrow PRUNE_SUBNODE$) and the process backtracks to a node at a higher level in the tree (lines 14 to 16). Otherwise, the whole process will go on ($status \leftarrow PROCESS_CONTINUE$).

¹This function is used to check the solution retrieved from s is similar with any solutions retrieved from states in Q or not.

² $lc = 0$ means the state s is not satisfied with the location constraint

Algorithm 1 Backtrack(s, Q, seq, A)

```
1:  $status \leftarrow PROCESS\_CONTINUE$ ;
2: if the size of  $Q$  is equal to  $k$  then
3:    $status \leftarrow PROCESS\_FINISH$ ;
4: end if
5: if  $status = PROCESS\_CONTINUE$  then
6:    $isPrune \leftarrow true$ ;
7:   if  $gc$  of the  $\omega$  of  $s$  is larger than  $gc$  in  $\omega_g$  then
8:      $isPrune \leftarrow false$ ;
9:   end if
10:  if  $lc$  of the  $\omega$  of  $s$  is equal to 0 then
11:     $isPrune \leftarrow true$ ;
12:  end if
13:  if  $isPrune$  is false then
14:    if  $\omega$  of  $s$  is greater than or equal to  $\omega_g$  and  $checkNoSim(Q, s)^1$  then
15:      Push  $s$  into  $Q$ ;
16:       $status \leftarrow PRUNE\_SUBNODE$ ;
17:    else
18:       $status \leftarrow PROCESS\_CONTINUE$ ;
19:    end if
20:  else
21:     $status \leftarrow PRUNE\_SUBNODE$ ;
22:  end if
23: end if
24: if  $status = PROCESS\_CONTINUE$  then
25:    $applicableServices \leftarrow ActFilter(s, A)$ ;
26:   for  $\forall service$  in  $applicableServices$  do
27:      $s' \leftarrow \gamma(s, service, seq)$ ;
28:      $Backtrack(s', Q, seq + 1, applicableServices)$ ;
29:   end for
30: else
31:   return;
32: end if
```

Algorithm 2 checkNoSim(Q, s)

```
1: noSim  $\leftarrow$  true;
2: for  $\forall s1$  in Q do
3:   Calculate the Levenshtein distance ld between the two solutions retrieved from
   s, s1;
4:   Calculate the number of services n in the solution retrieved from s;
5:   Calculate the number of services n1 in the solution retrieved from s1;
6:   Calculate similarity sim  $\leftarrow 1 - \frac{ld}{\max(n, n1)}$ ;
7:   if sim  $\geq$  0.5 then
8:     noSim  $\leftarrow$  false;
9:     break;
10:  end if
11: end for
12: return noSim;
```

*ActFilter*³(Line 25) is used to retrieve the applicable services from *A* depending on the current state's circumstance, which follows the steps:

1. We check if the *comp* of the current state's ω is equal to 1. If so, this means the *activityTypeExp* of the current state has already contained all service types that the user wants to take and no more services are applicable. Otherwise, we need to choose applicable services from *A*.
2. We check the types of service. If one type of services has been already contained by *activityTypeExp* of the current state, this type of activity is not applicable. After this check, a part of the services are filtered. The rest of the services will do the next check based on their start locations;
3. We check the start locations of those services. If one service's start location is equal to the location of the current state, this service is applicable. After this check, all of the residual services are applicable.

After that, we use *service* in *AvailableServices* and γ to generate new node *s'* (line 27). Then, we call the Backtrack procedure again to run the next iteration(line 28).

Algorithm 2 provide one process to ensure the diversity of solutions. First, we calculate the Levenshtein distance [Wik14b] between the solution from the current

³The detail is not included in the manuscript due to its triviality.

state and solutions from existing states in Q , because each solution could be regarded as an ordered service sequence. In information theory and computer science, the Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the minimum number of single-character edits (*i.e.*, insertions, deletions or substitutions) required to change one word into the other. Mathematically, the Levenshtein distance between two strings a, b , which can be seen as two character sequences, is given by $lev_{a,b}(|a|, |b|)$ [Wik14b] where

$$lev_{a,b}(i, j) = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0 \\ \min \begin{cases} lev_{a,b}(i-1, j) + 1 \\ lev_{a,b}(i, j-1) + 1 \\ lev_{a,b}(i-1, j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \quad (21)$$

Therefore, we could use the same method to calculate the Levenshtein distance between two solutions which are regarded as two service sequences. After generating Levenshtein distances, we use those Levenshtein distances to produce similarities. If there is one similarity which is greater than or equal to 0.5, we can treat the solution from the current state is similar with the solution from one of the existing states in Q . Because we retrieve one solution from one state, we say the current state is similar with one of the existing states in Q at this time. Then, in order to guarantee the solutions' diversity, a "false" indicator will be returned and the solution from this state will be discarded. This strategy has the possibility to miss the best solution.

Algorithm 1 terminates after k solutions are reported or no states expand. To reduce the space requirements, our algorithm uses the depth-first search(DFS) as the search strategy. However, using DFS has the possibility to miss the best solution because DFS always selects the most recently generated node or the deepest node to expand next [Zha96]. For instance, there is more than one goal node in the tree we build, and our search decided to first expand the first subtree of the root where there is a solution at a very deep level of this subtree, at the same time the other one subtree of the root has the best solution, here comes the non-optimality of DFS that it is not guaranteed that the first goal to find is the optimal one, so we can conclude that DFS is not optimal [RNC⁺95].

⁴ $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and equal to 1 otherwise.

Theorem 1 *The time complexity of backtracking algorithm is exponential to the branching factor b and the maximum depth m .*

Proof: b is the branching factor which means the number of children at each node and m is the maximum depth of any path in the search tree. The depth-first search is asymptotically optimal because most nodes will not have a child-node which has the same ω value. Hence, the expected number of nodes expanded by the depth-first search for finding several optimal states of a tree $T(b, m)$, as $m \rightarrow \infty$, is $\theta(\beta^m)$, where β is a constant, $1 < \beta < b$ [Zha96]. \square

5.3 Re-planning

After one user completes the service composition, the user can get several solutions. However, he or she may not be satisfied with those solutions. It is better to let users replan to have new solutions. As we talk about the service composition problem and algorithm in above sections, we can see one solution is generated by adopting different services. Therefore, the critical point is changing solutions is to restrict or change the set of available services. In this thesis, we use that strategy to replan. We provide three options to users,

- Editing the original user's query to completely replan;
- Use the original user's query to replan, but ignore all the services found at the first time;
- Use the original user's query to replan and keep some specific services in the existing solutions (the user can choose the services);

The replanning process is shown in Figure 12. The replanning request includes users' inputs(query) and the data of activities in existing solutions. We use the function in Section 4.3 to collect business services. After that, the service processing part will dispose those services based on users' options. When all the qualified services are ready, we invoke the direction function to discover the directions between each of the two services. Finally, we pass those services to the composition engine to build new solutions.

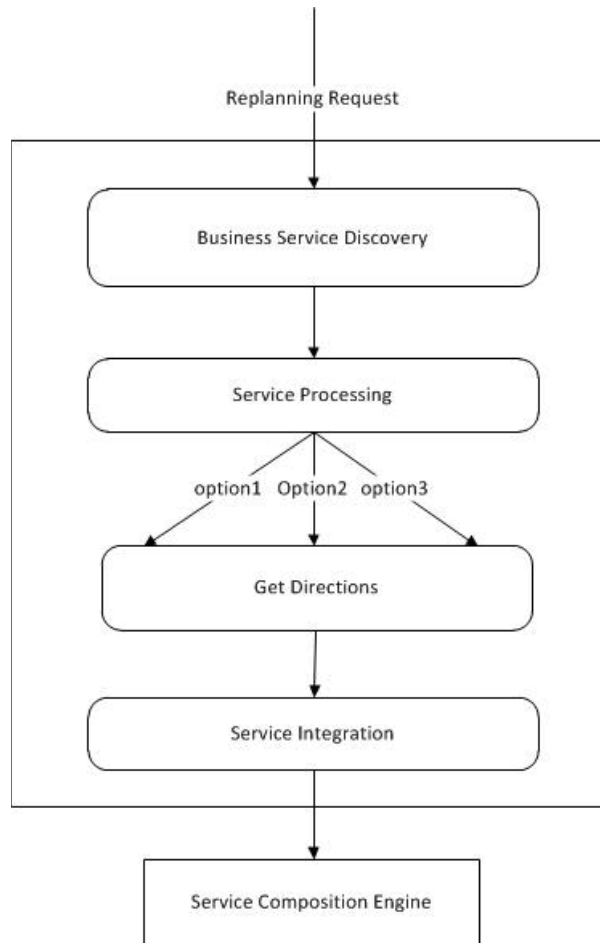


Figure 12: Re-planning Process

5.4 Limitation

In this thesis, we propose the models for contexts and services, and the constraints which begin with the scenario we interest. Moreover, we also create an algorithm features those constraints which has an ability to adopt the models for contexts and services. These models, constraints and the algorithm are able to work well for the scenarios which are similar with the scenario we used as the motivation, like making a plan for visiting an amusement park in a short time period because there are many choices for users to choose in an amusement park. However, there are several limitations for those models, constraints and the algorithm if we want to extend their usage scope. The limitations of them are as below.

1. Models for context and service

The context model we propose could be considered as key-value pairs plus predicate logic expressions and a domain independent ontology. All the types in this ontology are domain independent. But the subtypes we defined cannot cover all the general concepts in our daily life. For example, the subtypes of type “service” are domain dependent. If people want to use the model in some special scenarios, they may need to do some customizations.

2. Constraints

In this thesis, the constraints are used to evaluate contexts. These constraints mainly focus on the time consumption, money cost and location. If the time cost, the money cost and location are not key restrictions in one scenario, those constraints are not efficient to do the evaluation. Additionally, we define a variable ω including all the values of constraints to indicate the quality of plans (solutions). The ω does not have the ability to work well for all kinds of scenarios. In some cases, the ω does not need to contain all kinds of constraints. For example, if one user does not care about the money cost of a plan, it is not necessary to include the value of cost constraint in the ω . Therefore, the ω need to have an ability to contain the appropriate constraints dynamically based on the scenario.

3. Service composition algorithm

We adopt a customized backtracking search algorithm as the service composition algorithm. In this algorithm, we also add several individual constraints to optimize the whole search process. Those individual constraints we employ will not be able to work efficiently in some scenarios because users may not care about the distance between the location of services and their current location. At that time, the location constraint we use is not appropriate. Moreover, the algorithm we use can be regarded as a single thread search process. We can also optimize this algorithm using multi-thread technology.

Chapter 6

Implementation of the Personal Entertainment Planner

In this chapter, we present the design and the implementation of the Personal Entertainment Planner as Web application.

6.1 Introduction

6.1.1 Goals and Objectives

The objective of this system is to produce an application as a prototype to demonstrate the feasibility of our procedure for doing context-aware non-electronic service discovery and mashup.

6.1.2 Statement of Scope

The implementation of the Personal Entertainment planner includes four parts which are a user view application, a service process application, a service discovery application and a database. The user view application is used to interacted with users. We do service composition and service mashup in the service process application. The service discovery application is only used to implement the service discovery process and the database stores the data related to the whole system.

6.1.3 Major Constraints

In order to run this application, the user must use a device to run a Web browser supporting HTML5, as well as having an ability to access the Internet.

6.2 Design Considerations

All development work for the entertainment planner is done in the Eclipse Integrated Development Environment(IDE) on Windows 7 machines with the Java Development Kit (JDK 1.7), Jersey (1.7) and Spring MVC (3.2.3). The whole system is deployed on a Centos 6.5 Linux machine we rent from Godaddy. The application server we use is Tomcat (7.0.52), and we employ Mysql (5.5) as our database to store all the data of our system. Testing of system was done via two kinds of Web browsers (Safari on IOS 7 and Chrome on Windows 7). HTML5 technologies are well-supported in those two browsers.

6.3 System Architecture

Figure 13 indicates the system's architecture. This system is composed of three Web applications: User View Application, Service Process Application and Service Discovery Application. All of those three applications are Java Web application. We wrap the service composition application and the service discovery application as RESTful services. Those applications use the HTTP protocol to communicate. The User View Application is responsible for collecting user's requirements, interacting with service process application and displaying the result to users. This application has two versions of UI: desktop and mobile. Changing the user interface depends on different kind of devices. The Service Process Application contains two components (service composition engine and service mashup engine), which is used to do service composition and service mashup. The main function of the Service Discovery Application is to collect services from public service providers and push those services to the service composition engine in Service Process Application. The service discovery application and mashup part in our system can be regarded as an information aggregation component. In this architecture, all the services we collect are stored in the database. The service discovery application will store services in the database after discovering

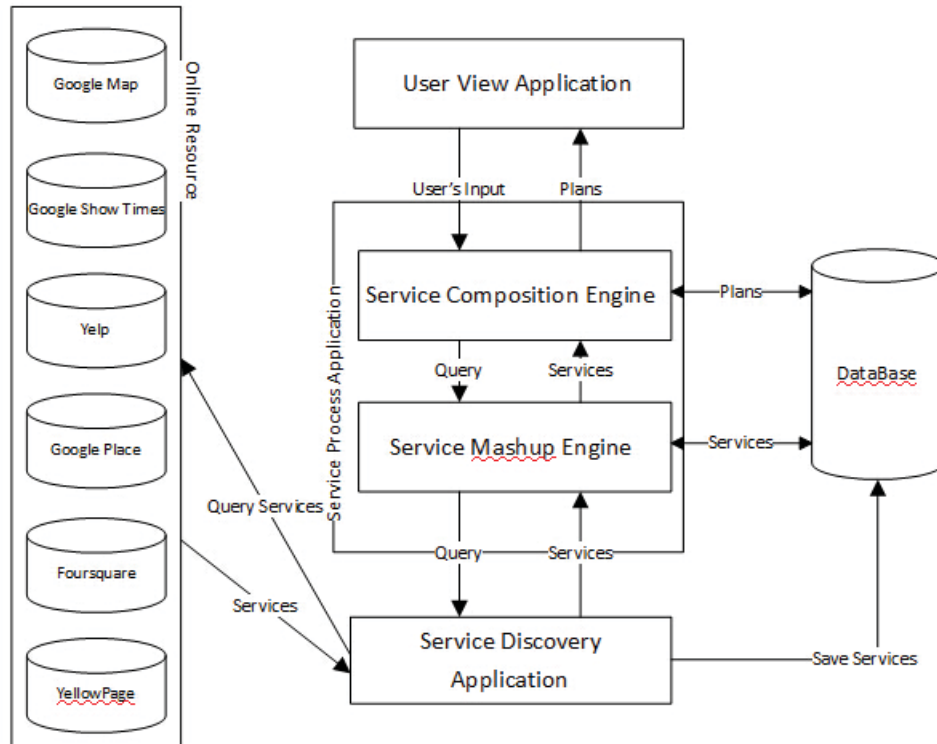


Figure 13: System Architecture

services. The mashup engine firstly queries the services from database depending on user inputs. If the result services we collect are not enough, the engine will send a request to the service discovery application to retrieve services.

This architecture divides the whole system into three components. Each of them are both independent and related. The service discovery application can be seen as a RESTful service which returns services to the requesters based on their criteria. The Service composition engine and the service mashup engine are two constituent parts of the service process application. The service composition engine can be regarded as a RESTful service for generating plans depending on the user's input. Moreover, the service mashup engine is a RESTful service client which has an ability to invoke the service from the service discovery application. So, the service process application is both a RESTful service and a RESTful service client. In addition, the User View Application is not only a Web application, which is used to display the UI and gather the user's input information, but also a RESTful service client to communicate with the other RESTful services (service process application). Therefore, those three components can be connected over HTTP protocol. Because of adopting

this architecture, this system is more maintainable and reusable and also hides the complexity of processing data from the users.

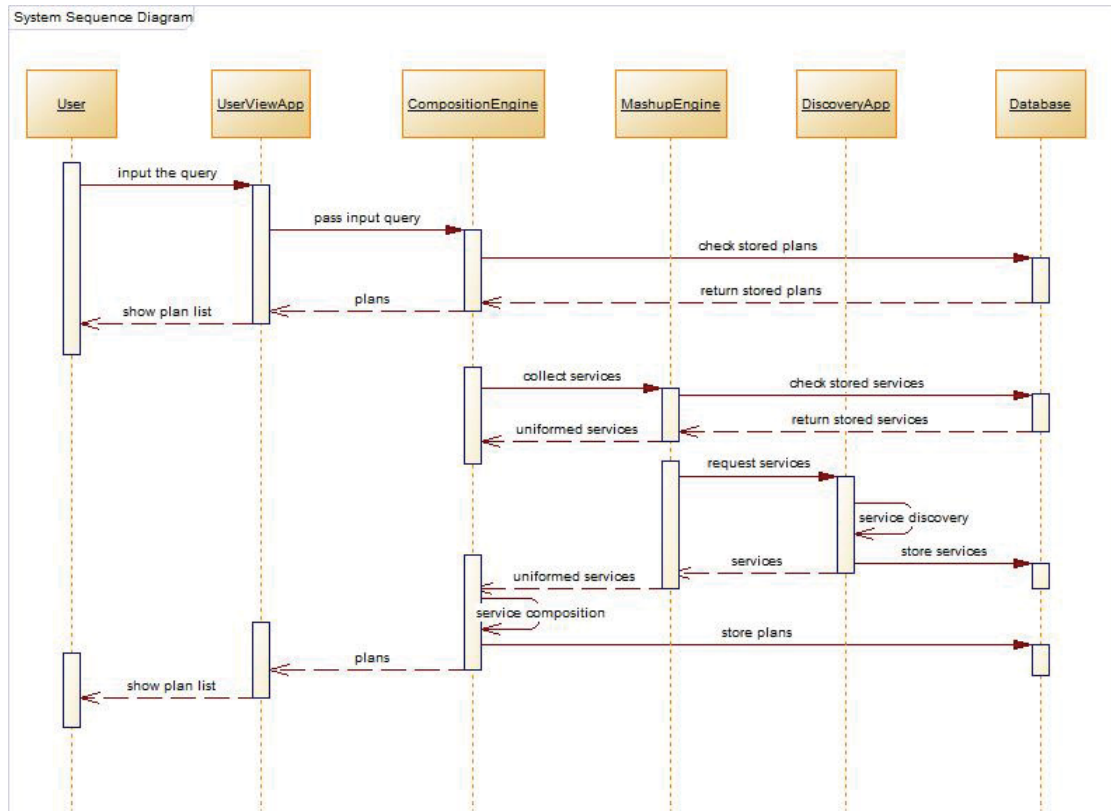


Figure 14: System Sequence Diagram

Those three applications of the system can be separated into four functional modules according to the system functionality, which are user view application, composition engine, mashup engine and discovery application. Users interact with the planner system through the user view application. Figure 14 shows the sequence diagram of the system. The user view application can be seen as an entry for the personal entertainment planner. When the personal entertainment planner is launched through a Web browser, the user is presented with the input UI provided by the user view application to input his query. Once the user's query is received by the user view application, the query will be passed to the composition engine. The composition engine will check the stored plans in the database based on the user's input. If there are some proper plans, the composition engine will fetch those plans and return them to the user view application in order to show the plan list. Otherwise, the composition engine should start to generate new plans for the user. At first, the composition

engine invokes the mashup engine to provide some activities (uniformed services) depending on the user's query. When the collecting service request is acquired by the mashup engine, it checks the database first. If there are no appropriate services, the mashup engine will call the service discovery application to capture services. After the service discovery process, the discovery application needs to store those services in the database and return these services to the mashup engine. Then, the mashup engine disposes those primitive services to the uniformed services and send them to the composition engine. When the composition engine obtains the uniformed services from the mashup engine, it generates plans using planning based service composition algorithm. Once the result is ready, the composition engine will save those plans in the database and return them to the user view application. The user can view the summary and the details of any plan via the user view application.

In the following sections, we will present the details of those four modules: user view application, discovery application, mashup engine and composition engine.

6.4 User View Application

6.4.1 Design Constraints

The user view application has two kinds of UI, one is for desktop browsers, the other one is for browsers on mobile devices. The user interfaces for destop browsers are able to handle more complex operations and contain more information. On the contrary, the style of user interfaces compatible with mobile browsers is different. The user interfaces for mobile devices should be simple and intuitive so that the user can easily identify what options they currently have to progress. Moreover, the UI components have to be easy to click for users over a touch screen and are able to ensure the accuracy of users' operations.

6.4.2 Input UI

In the user view application, the *Input UI* is used to let users input their criteria for generating plans.

The input UI for desktop browsers is shown in Figure 15. As a default, this application detects the user's current location automatically. If the user wants to edit

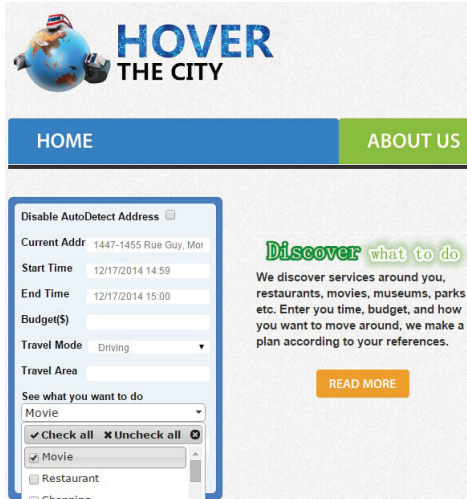


Figure 15: Input UI

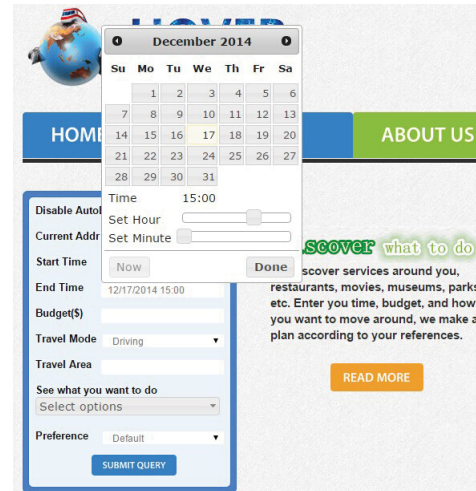


Figure 16: Setting Time

the address, he should check the checkbox “Disable AutoDetect Address”. The user can select the start time point or the end time point through a dialog as presented in Figure 16. “Budget” means how much money the user wants to spend on entertainment and the unit is in canadian dollars. When the user clicks the input area of “Travel Area”, he can choose the area he want to go through a widget as shown in Figure 17. This UI also provides a list for users to choose several kinds of activities (see Figure 15).

We present the input UI for mobile browsers in Figure 18. Most functions in this mobile input UI are the same as the UI for desktop browsers. However, we change the method for choosing the “travel area”. Unlike the UI for desktop browsers, when one user touches the input area of “Travel Area”, he will go to the other page (see Figure 19) to choose the place and the range. After choosing the area, the user will return to the input UI because the screen size of mobile devices is too small to use a dialog widget for selecting a specific area on mobile devices.

6.4.3 Result List UI

After the user query is submitted, the user view application provides a set of UI to present the result. Our principle for designing this UI is to display the results information for users as much as we can. The result list UI is able to show the summary and the detail of each plan (see Figure 20 and Figure 23). The user can also check the detail of each activity from the UI we provide, for example, Figure 21

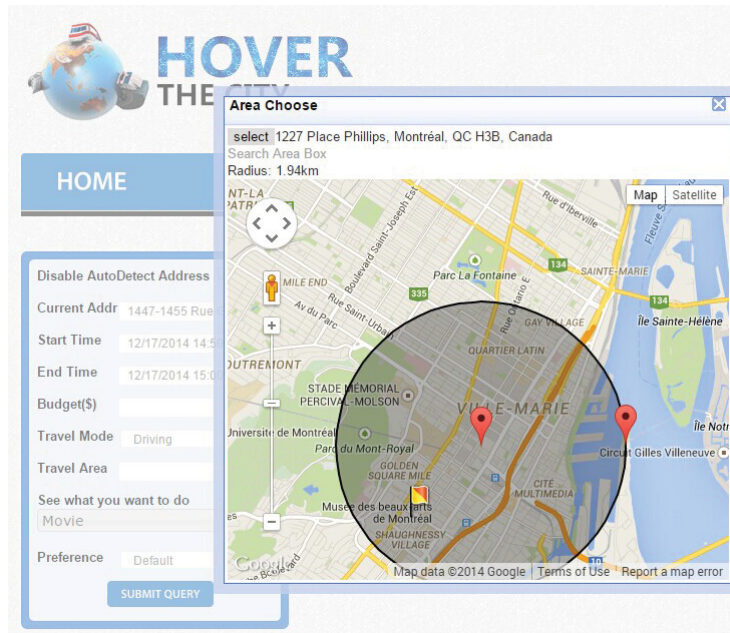


Figure 17: Choosing Travel Area

shows the detail of a restaurant. Moreover, we design a section on the result list UI for users to replan (see Figure 22 and Figure 24).

6.5 Service Discovery Application

The service discovery application is only used for doing service discovery. After finishing the process of service discovery, the application will return services to the requester. This application is able to provide three kinds of services based on the way of retrieving them, which are the direction services, general business services¹ and movie services.

6.5.1 Application Architecture

Figure 25 is the architecture of this application. As Figure 25 shows, the application composed of the following three parts.

- **ServiceProvider** provides an entry for users' requests. It can invoke the

¹General business services in this thesis are used to represent the non-electronic services except direction services and movie services, because direction services need to use another services as the input and movie services come from a html service.

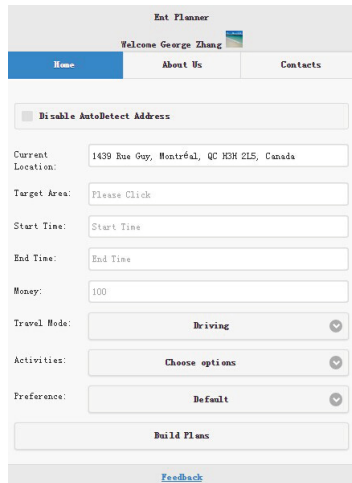


Figure 18: Input UI (Mobile)

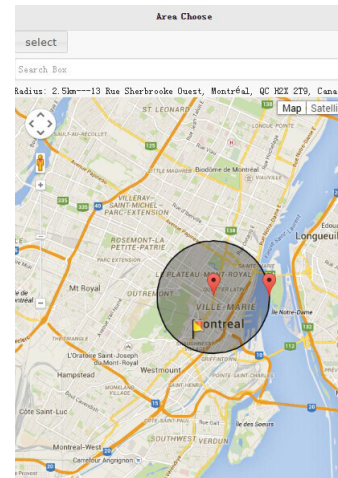


Figure 19: Choosing Travel Area (Mobile)



Figure 20: Result List UI (Desktop)

different kind of discovery services based on one user's request and return the result services. For example, if one user's request is for collecting movies, the `ServiceProvider` will invoke the `MovieProvider` to gather the information about movies. Currently, the `ServiceProvider` provides three options for users, which are "movie", "direction", and "general service".

- The `Discovery Services` part includes three providers: `DirectionProvider`, `MovieProvider`, `GeneralServiceProvider`. Each provider is independent. One provider is employed to call one or several general service processors to gather services depending on the user's request. This provider will save the services to the database and return those services to the `ServiceProvider`.

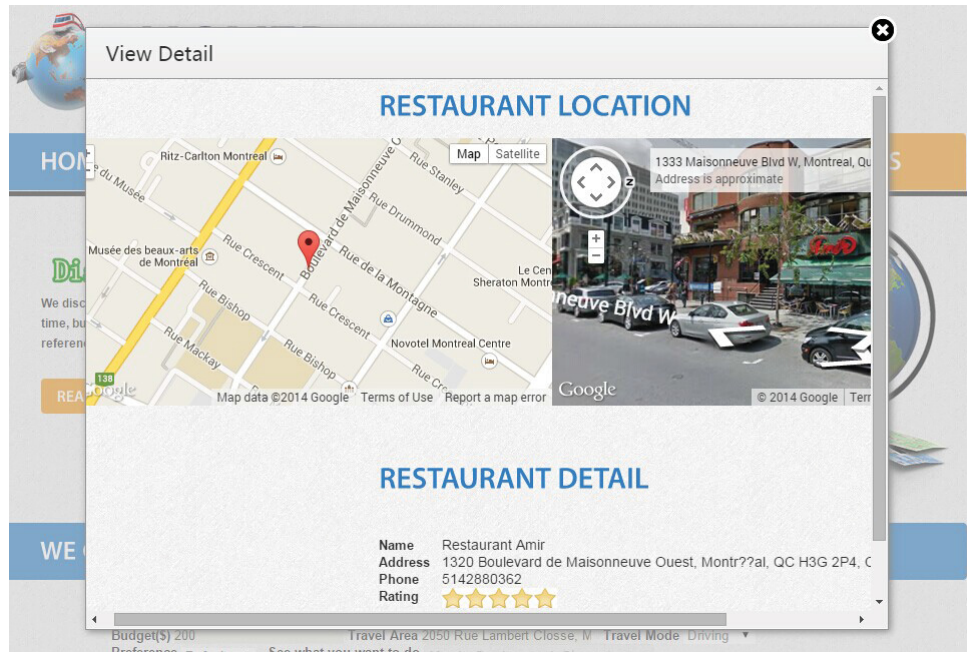


Figure 21: Restaurant Detail

- Currently, the **General Service Processors** part contains six service processors. Each processor is responsible for searching services from one general searching engine service, *e.g.*, Yelp, Google Place and Foursquare.

6.5.2 Service Discovery Implementation

We implemented the service discovery application based on its architecture. In other word, this means three parts should be implemented.

1. **ServiceProvider** is implemented as a resource class to provide RESTful services by using Jersey. It can handle three kinds of users' requests based on the functions of this application. Those requests include "movie request", "direction request", "general service request". For each kind of request, the **ServiceProvider** offers one entry.
2. We created three individual classes to implement the **Discovery Services** part, which are **DirectionProvider**, **MovieProvider**, **GeneralServiceProvider**. Those three classes are invoked by the **ServiceProvider** directly. **DirectionProvider** will call "Google Map processor" to generate the direction services. **MovieProvider**

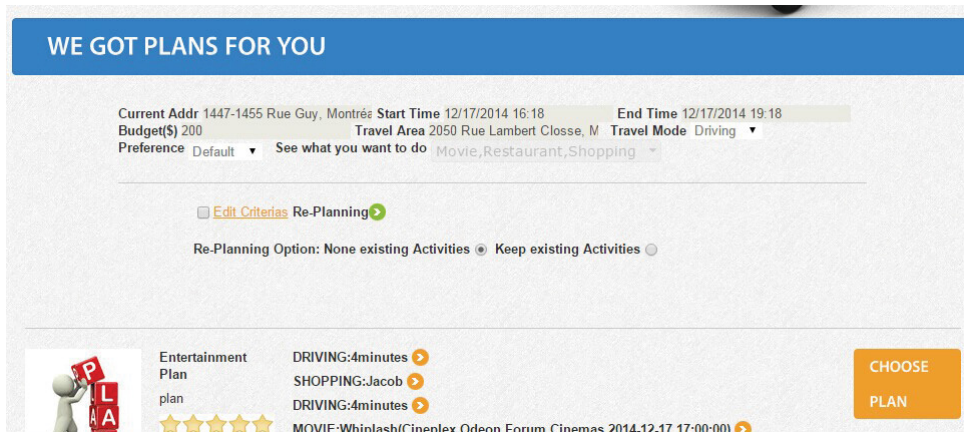


Figure 22: Re-planning Part (Desktop)

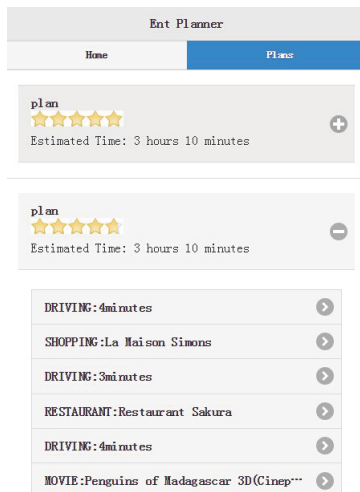


Figure 23: Result List UI (Mobile)

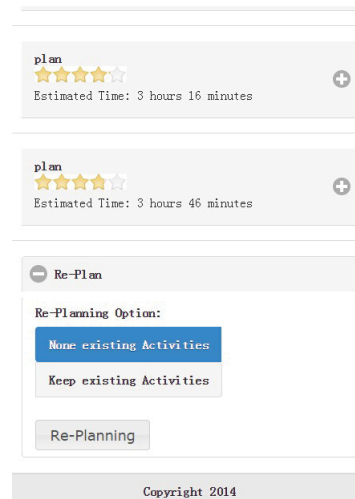


Figure 24: Re-planning Part (Mobile)

will call “Google Show Time processor” to retrieve the movie information. The `GeneralServiceProvider` is different than the other two. It should invoke four processors to dispose one request. After that, an integration operation needs to be done to unify the style of services from various processors. Therefore, we use multithreads to invoke the four processors in order to save time cost. Moreover, we add one function in the `GeneralServiceProvider` to make those services be uniform.

3. The `General Service Processors` part can be implemented by building six processor classes on the basis of the application’s architecture. We can classify those processors as two categories, RESTful service processors (“Google Map”,

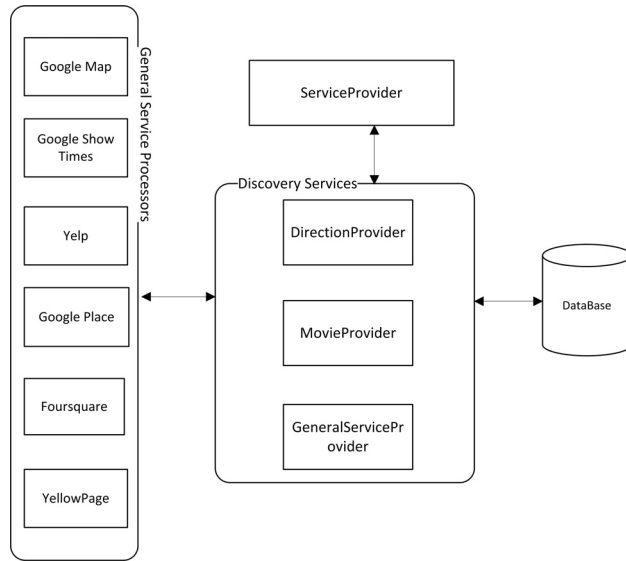


Figure 25: Architecture of Service Discovery Application

“Yelp”, “Google Place”, “Foursquare” and “YellowPage”) and the HTML service processor (“Google Show Times”). The RESTful services we adopted in this application all support JSON result. The key point for RESTful service processors is how to convert the JSON strings to Java Objects. For this application, we choose Google Gson. Figure 26 indicates the whole executive procedure of RESTful service processors. For the HTML service processor, the main task is to parse the HTML pages. In this work, we use the Jsoup library to invoke and parse those HTML pages to retrieve the movies’ information. Then we are able to build movie services based on those movies’ information.



Figure 26: RESTful Service

6.6 Service Mashup Engine

6.6.1 Building Models

As Figure 27 shows, we use the following objects to model the service mashup engine.

- `ServiceParser` is an interface, which defines two abstract methods named

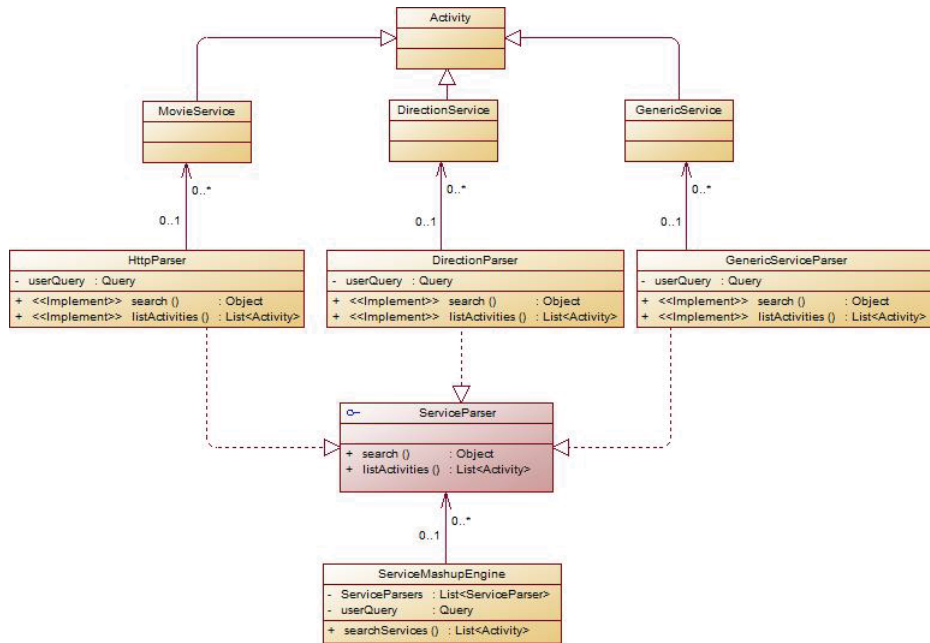


Figure 27: Service Mashup Engine Class Diagram

`search` and `listActivities` respectively that can be implemented by other classes. The `search` is used to collect services from the database or service discovery application. After that, the `listActivities` method can convert those services to activities. `HttpParser`, `DirectionParser`, `GenericServiceParser` implement `ServiceParser`, gather different kinds of service, and return a list result containing `Activity`.

- `ServiceMashupEngine` is a controller class in the mashup engine. The Service composition engine invokes `ServiceMashupEngine` class to retrieve services. After receiving one request from service composition engine, the `ServiceMashupEngine` is able to call several `ServiceParsers` to gain the desired services and return those services.
- `Activity` is an entity class to represent the *Service* defined in Definition 4 in Chapter 3.
- `MovieService`, `DirectionService` and `GenericService` extend `Activity` class and represent a specific kind of activity separately.
- `HttpParser` is used to collect movie services from the database or the service

discovery application and convert the result into a `MovieService` list.

- `DirectionParser` is used to collect direction services from the database or the service discovery application and convert the result into a `DirectionService` list.
- `GenericServiceParser` is used to collect general services from the database or the service discovery application and convert the result into a `GenericService` list.

6.6.2 Service Mashup Engine Implementation

According to our model for the service mashup engine, the objects that need to be implemented can be divided into three kinds, which are entity classes, parsers, and one controller class.

1. `Activity`, `MovieService`, `DirectionService` and `GenericService` are general entity classes. We use those classes to represent entities and store information.
2. In this part, we need to implement a set of parsers which realize the interface `ServiceParser`. All of those three parsers have to implement two main functions (query services from database and request services from the service discovery application). When the mashup engine invokes one parser to obtain services, the parser firstly will query services from the database. If there are no exist proper services, the parser will request services from the service discovery application.
3. `ServiceMashupEngine` can be regarded as a controller class in this part. It will invoke a list of parsers to gather services based on user's query from composition engine. Based on our design (see Figure 11 in Section 4.3), there is an order of execution for those parsers. At first, the `ServiceMashupEngine` should invoke parsers except `DirectionParser`. When the other parsers finish executing, the `DirectionParser` will be invoked. To ensure the efficiency of the process, we use multithreading to implement this class. For example, one `HttpParser` and several `GenericServiceParsers` can start running simultaneously.

6.7 Service Composition

6.7.1 Building Models

In order to implement service composition, we model the service composition as shown in Figure 28. The Web service composition problem can be regarded as a planning problem and the Backtracking Search Algorithm is employed as our planning algorithm in our work. Because we adopt the object-oriented method, we define the following objects in the implementation of the BSA using Java language.

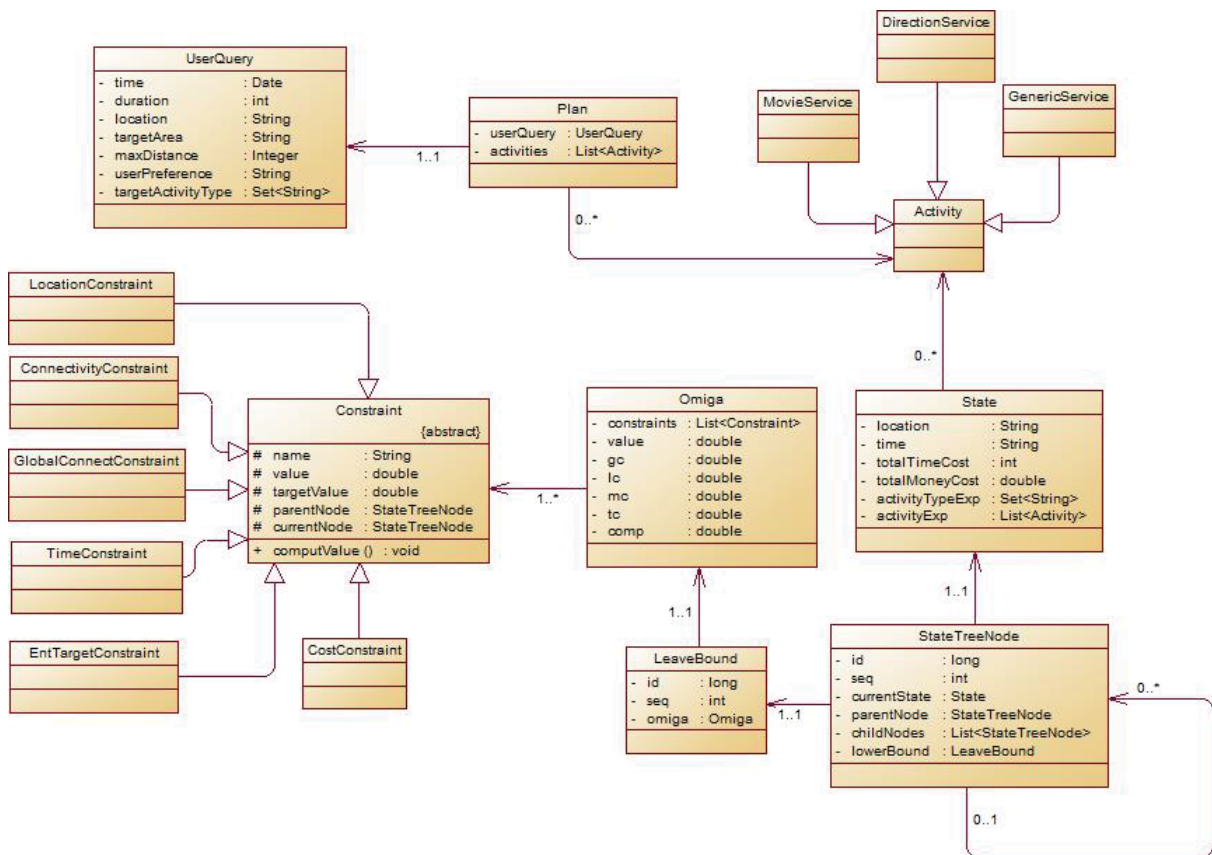


Figure 28: Service Mashup Engine Class Diagram

- **Constraint**

Constraint is an abstract class to define the common properties and abstract operations for other specific constraints. The common properties of this class includes *name*, *value*, *targetValue*, *parentNode* and *currentNode*. *targetValue*, *parentNode* and *currentNode* are used to calculate the value of one specific

constraint. `Constraint` also contains an abstract operation *computeValue* which the specific constraints have to implement for calculating the value of constraints.

- `LocationConstraint`, `ConnectivityConstraint`, `GlobalConnectConstraint`, `TimeConstraint`, `EntTargetConstraint` and `CostConstraint`

`LocationConstraint`, `ConnectivityConstraint`, `GlobalConnectConstraint`, `TimeConstraint`, `EntTargetConstraint` and `CostConstraint` are subclasses of class `Constraint` to represent each specific type of `Constraint`. Those classes are implemented based on the definitions in Section 3.3.

- `LocationConstraint` is used to represent the constraint *locationConstraint*. The `computeValue` function is implemented depending on the calculation method of the constraint *locationConstraint*.
- `ConnectivityConstraint` is used to represent the constraint *connectConstraint*. The `computeValue` function is implemented depending on the calculation method of the constraint *connectConstraint*.
- `GlobalConnectConstraint` is used to represent the constraint *globalConnect*. The `computeValue` function is implemented based on the calculation method of the constraint *globalConnect* (see Equation 18).
- `TimeConstraint` is used to represent the constraint *timeConstraint*. The `computeValue` function is implemented depending on the calculation method of the constraint *timeConstraint*.
- `EntTargetConstraint` is used to represent the constraint *completeness*. The `computeValue` function is implemented according to the calculation method of the constraint *completeness* (see Equation 11).
- `CostConstraint` is used to represent the constraint *costConstraint*. The `computeValue` function is implemented based on the calculation method of the constraint *costConstraint*.

- **State**

As described in Definition 14, a `State` object is a context. *i.e.*, a set of variable assignments, including user's inputs variables and circumstance variables.

- **Omega**

In our work, class **Omega** is created according to the definition of ω (see Definition 12). A list of **Constraint**'s instances, which are employed to assess one instance of **State**, will be included in the object **Omega** as property **constraints**. Therefore, we can initialize one **Omega** instance by using **constraints** and one instance of **Omega** can be adopted to evaluate an instance of **State**. The property **value** in **Omega** represents the value of ω and it can be computed based on Definition 13.

- **LeaveBound**

LeaveBound object is used to contain the ω and the information of one node, it has three properties, *i.e.*, **id**, **seq** and **omega**. The values of properties **id** and **seq** represent the information of **StateTreeNode** which **LeaveBound** object associates to. The property of **omega** is an instance of **Omega**. Thus, one instance of **LeaveBound** can also be applied to evaluate an instance of **State**. Furthermore, by comparing with the value of **omega**, we can compare two instances of **LeaveBound**. The higher the value of **omega** is, the better the leavebound is.

- **StateTreeNode**

In our work, backtracking search algorithm employs Depth-First search to build its search tree. For each node in the search tree, we define a object **StateTreeNode**, which contains a variable of **StateTreeNode** object to refer to its parent node and a list of **StateTreeNode** as its child nodes. One instance of **StateTreeNode** corresponds to one instance of **State** as the property **currentState**. Property **leaveBound** is used to indicate the quality of the **currentState**. In fact, a **StateTreeNode** object wraps an object of **State**.

- **UserQuery**

As defined in Definition 7, **UserQuery** object is described as a query sent by a user through the user view application.

- **Activity**

In according to the Definition 4, we define an **Activity** object to represent the concept of **service**.

- `MovieService`, `DirectionService` and `GenericService`

`MovieService`, `DirectionService` and `GenericService` are sub-classes of class `Activity` to represent each specific type of `Activity`.

- `Plan`

`Plan` object represents one plan returned by the planner as in Definition 19, which contains a list of activities and an instance of `UserQuery`.

6.7.2 Service Composition Implementation

We proposed a customized backtracking search algorithm (Algorithm 1) for service composition described in Chapter 5. In this part, we utilize the Algorithm 1 as an example to illustrate our service composition algorithm implementation. Suppose a user’s inputs are as in Figure 29. The user wants to go somewhere within 2 kilometers around the target area where the geological location is (45.499857, -73.573546) to spend three hours from 14:02 to 17:02. Currently, the user located in the area, identified as the geological location (45.492219, -73.581892). During the three hours, the user would like to spend about 100 dollars in watching movie, doing shopping and having meal. The user’s preference is “balanced”.

We build a search tree to implement the algorithm. During the process of building the search tree, those constraints are used as a guide. Actually, the search tree beginning from the initial state is assumed to go to a series of current state and finally generate several goal states. In the following, we illustrate the method of building the search tree and the state change during the tree-building phase.

1. Based on the inputs through the UI, we can build the initial state s_0 and the goal ω ω_g shown as below.

$$s_0 = \langle activityTypeExp = \{\}, activityExp = \{\}, totalTimeCost = 0 \\ totalMoneyCost = 0, location = "45.492219, -73.581892", time = 14 : 02 \rangle \quad (22)$$

$$\omega_g = \langle lc = 1, comp = 1, gc = 0.95, tc = 0.85, mc = 0.85 \rangle \quad (23)$$

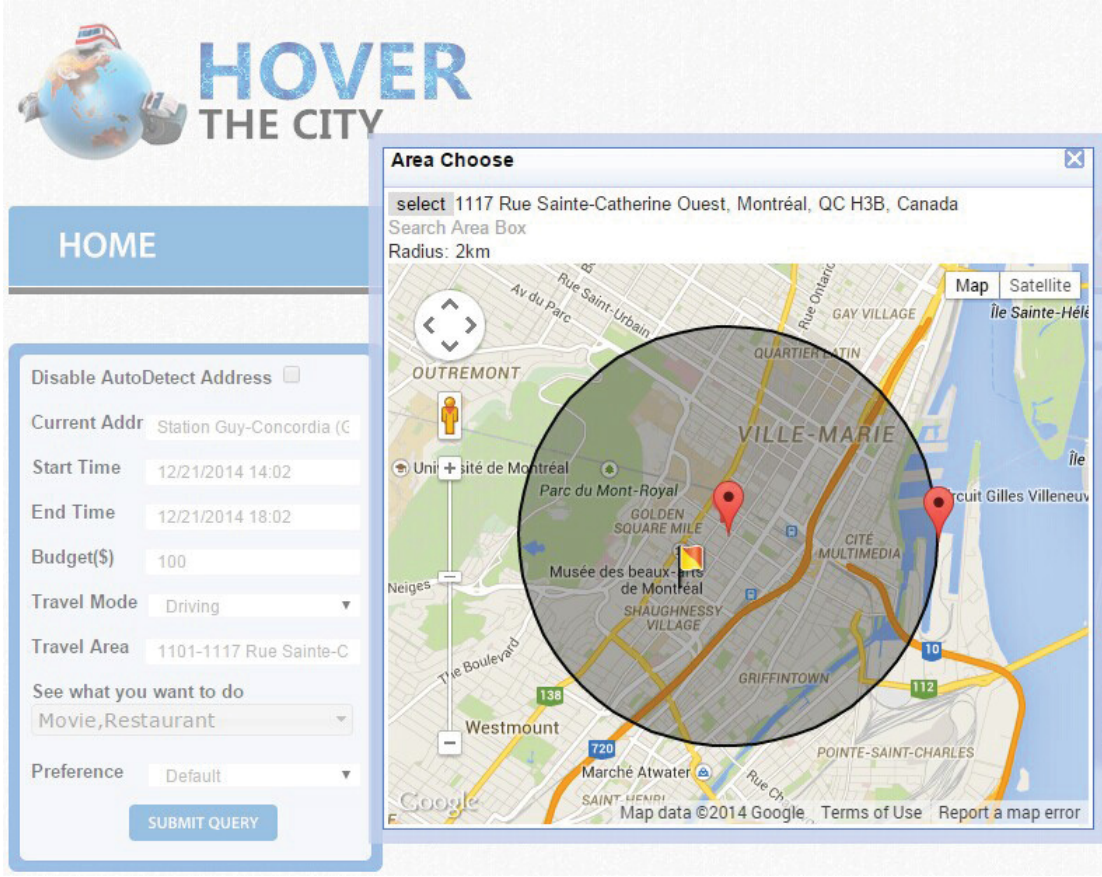


Figure 29: User Inputs Sample

According to the detail of initial state s_0 , we can also create an ω_0 and a LB_0 for s_0 , *i.e.*, $\omega_0 = \langle lc = 1, comp = 0, gc = 1, tc = 0, mc = 0 \rangle$. Q is a priority queue to store qualified states. Moreover, we use a variable *status* to represent the status of the whole procedure ($status \leftarrow PROCESS_CONTINUE$).

2. Starting algorithm from checking the Q . If the size of Q is equal to the k^2 , the whole process will be finished ($status \leftarrow PROCESS_FINISH$).
3. Next, if *status* equals to $PROCESS_CONTINUE$, the current state s in the process will be evaluated. If $\omega.gc$ of LB related to s is smaller than $\omega.gc$ of ω_g , s will be pruned. If $\omega.lc$ of LB related to s is smaller than 1, s will be pruned either. Otherwise, s can be saw as a candidate state.
4. if s is a candidate state, we will check if this state can be insert into the Q or

² $k=5$, because this system returns 5 plans once by default.

not. If so, this state will be inserted into the Q and we will prune the subtree of this state node($status \leftarrow PRUNE_SUBNODE$). Otherwise, this means its child states still have the possibility to be solutions, continuing the process.

5. After the checking procedure for the state, if the the current state s is a candidate state, we are able to use this state s to generate new states by applying activities. In order to retrieve a list of applicable activities for state s , we use a function named *ActFilter* to check all the activities in activity set A . The function *ActFilter* can return a list of applicable activities, which follows the steps:
 - (a) First we check if the *comp* of the current state's ω is equal to 1. If so, this means the *activityTypeExp* of the current state has already contained all service types that the user wants to take and no more services are applicable. Otherwise, we need to choose applicable services from A .
 - (b) Then we check the types of service. If one type of services has been already contained by *activityTypeExp* of the current state, this type of activities is not applicable. After this check, a part of services are filtered. The rest of services will do the next check based on their start locations;
 - (c) Lastly we check the start locations of those services. If one service's start location is equal to the location of the current state, this service is applicable. After this check, all of the residual services are applicable.
6. After retrieving a list of applicable activities, we continue to traverse the activity list. The state s_0 will apply activities in activity list one by one. We pick up a random activity a_1 as an example. a_1 is a direction activity. After applying a_1 to s_0 , a new state s_1 is added to the search tree, *i.e.*, $s_1 = \langle activityTypeExp = \{“direction”\}, activityExp = \{a_1\}, totalTimeCost = 3, totoalMoneyCost = 10, location = “45.500907, -73.571976”, time = 14 : 05 \rangle$. We can also build the ω and leave bound for the state s_1 , *i.e.*, $\omega_1 = \langle lc = 1, comp = 0, gc = 1, tc = 0.125, mc = 0.1 \rangle$.
7. Then we use the state s_1 as the state parameter to call the backtracking procedure recursively in order to do the depth-first search(DFS).

8. The whole process will terminate after k solutions are reported or there are no states to expand. In other words, the whole process can end when the Q fulfills with the conditions defined at Step 2 or there is no more states can be generated. Figure 30 describes the whole process.

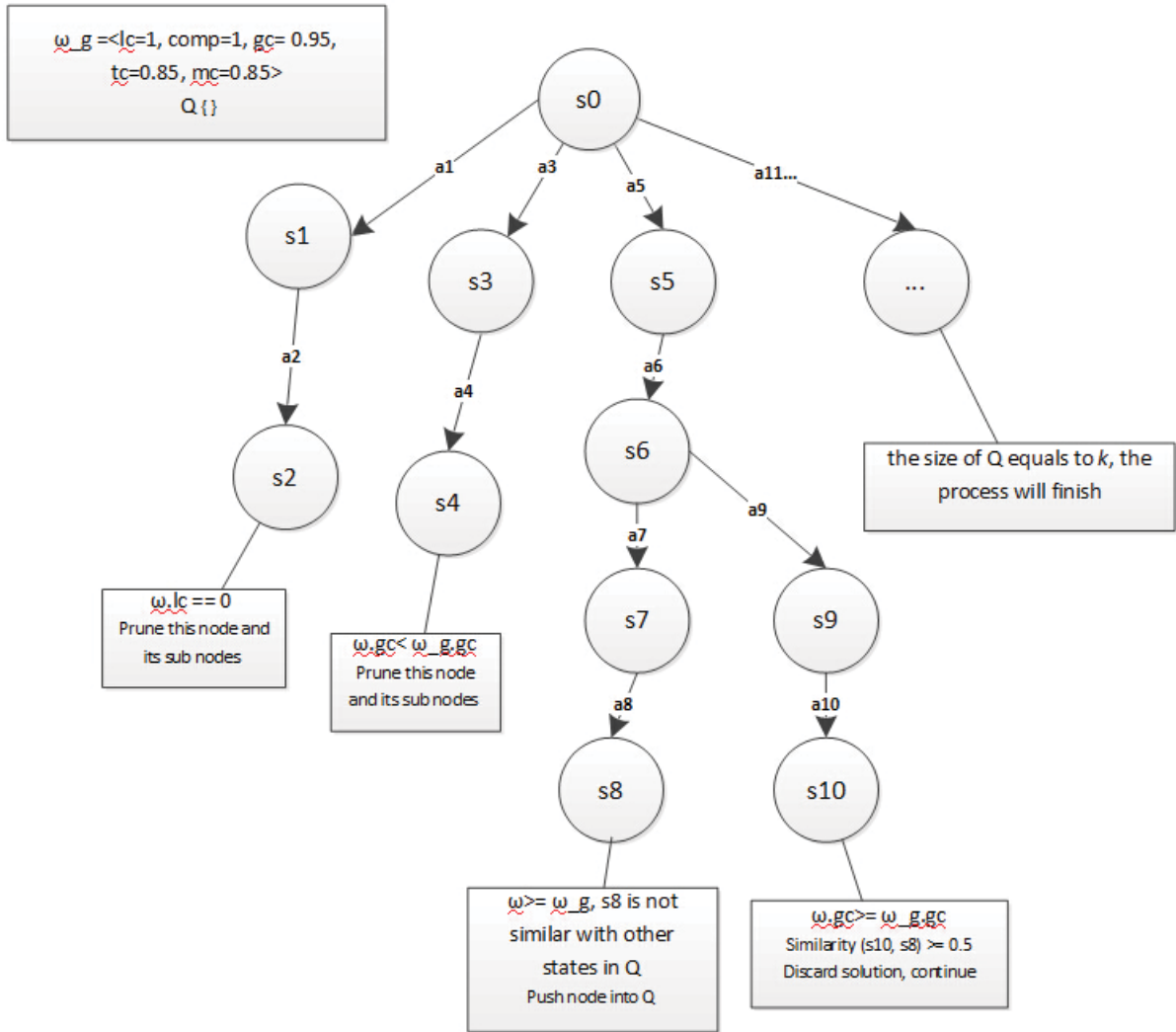


Figure 30: Process for the Algorithm

9. At last, we can obtain several states (nodes) from the Q . Every path in the search tree backward from those nodes to the root node is one of the solution. And all activities applied to the states in one path can build a plan which is what we recommend the user to do during that time.

6.8 Re-planning Implementation

According to the design of re-planning function in Section 5.3, the key point to change solutions is to restrict or change the set of available services (activities). The function of restricting or changing activities can be implemented in the mashup engine based on the system's architecture. In this system, we provide three re-planning options for users (see Section 5.3). Because of this, we define two new object for the implementation.

- **ReplanQuery**

ReplanQuery object is described as a query to replan sent by a user through the user view application. **ReplanQuery** class can be saw as a sub-class of **UserQuery**. It has three new properties “activities”, “activitiesKeep”, “random”. The property “activities” is a set which contains all the activities in the existing plans. The property “activitiesKeep” is a set which contains the activities in the existing plans. Those activities will appear in the new plans. The property “random” is boolean value. “random” is true means the new plans should not include any activity in the existing plans. Otherwise, the new plans will contain the activities in “activitiesKeep”.

- **ReplanMashupEngine**

ReplanMashupEngine is also a controller class in the mashup engine part. It has all the functions that the **ServiceMashupEngine** object has (see Section 6.6). However, the **ReplanMashupEngine** is used to handle the **ReplanQuery** from the composition engine. Moreover, it has functions to confine and change the activities.

The implementation of **ReplanMashupEngine** should be added into the service mashup part. The service mashup engine class diagram will be changed to Figure 31. In addition, we should also add support in service composition part in order to handle the **ReplanQuery** object. If one user send a **ReplanQuery** through the User View Application, the composition engine ought to invoke the **ReplanMashupEngine** object instead of **ServiceMashupEngine** object in the Service Mashup Engine part.

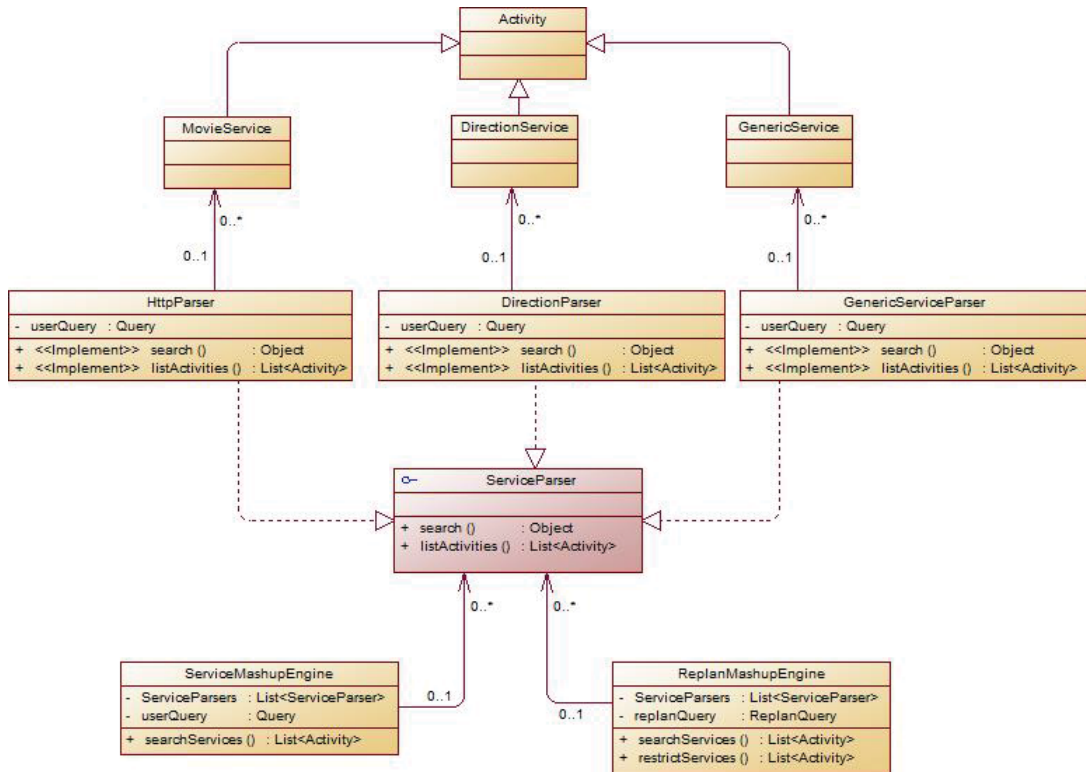


Figure 31: Service Mashup Engine Class Diagram (Re-planning)

6.9 Tests

We conduct tests to evaluate the effectiveness and performance of our approach that do non-electronic service composition and discovery. The objectives of our tests are to examine (1) if the time consumption of service discovery will not increase sharply with the rise of service types; (2) if time consumption of service composition is depended on the number of types of services we compose; (3) if users is satisfied with the experience of the whole mechanism.

6.9.1 Experiment Setup

- In our tests, we need to measure the time consumption. So, we have to find a way to calculate the time consumption. In this case, we add some code into the existing code to log the start time point and end time point of one execution.
- To evaluate the user experience of the system, we recruited 8 users to participate in our experiment. Nielsen suggests that the best user study for gathering

qualitative measures should be involve three to five users [Nie00]. Four of the 8 subjects are very familiar with Internet, and the remainder 4 subjects have the basic knowledge about Internet.

6.9.2 Experiment Procedure

Evaluating the performance of service discovery and composition

To evaluate the processing time consumption of service discovery and composition, we conducted two experiments. The first experiment is used to record the time cost of service discovery & mashup from 1 type to 5 types. We use the second experiment to calculate the time consumption of service composition from 1 type to 5 types. We adopt the same user query for both of two experiments. Before doing the first experiment, we empty the database to ensure the objectivity. The second experiment uses the output of the first experiment (activities) as one of its inputs.

The user query used for our experiments is as below:

$$\begin{aligned}
 userQuery = \langle &duration = 240, time = 14 : 00, \\
 &budget = 100, location = "1455 Guy Street, Montreal", \\
 &targetArea = "H3H2N7", userPreference = "balanced", \\
 &maxDistance = "1.5km", travelMode = "driving" \rangle
 \end{aligned}
 \tag{24}$$

The first experiment includes 5 test cases. The first one discovers one type of service (“movie”), the second one discover two types of service (“movie”, “restaurant”), the third one discovers three types of service (“movie”, “restaurant”, “shopping”), the forth one discovers four types of service (“movie”, “restaurant”, “shopping”, “museum”) and the last one discovers five types of service (“movie”, “restaurant”, “shopping”, “museum”, “park”). The second experiment also has five test cases. Each test case corresponds to one to five types of services respectively. In order to make the tests more impartial, we check all the possible combinations of types in each test case. For example, test case 1 corresponds to one type of services, which has five possibilities. Test case 2 corresponds to two types of services, which has ten possibilities.

Evaluating the user experience of the whole system

To evaluate the user satisfaction of the whole mechanism, we conducted a user study. We ask each of 8 subjects as described in Section 6.9.1 to build entertainment plans using our prototype. After the subjects completed the process for generating plans. we ask them to complete a short survey to access their experience. The survey contains the following questions:

- Are you satisfied with the response time of this application to do planning or re-planning? (Satisfaction of response time)
- How often did this application can give you a plan(s)? (Success rate of generating plans)
- Are you satisfied with the results generated by this application after planning or re-planning? (Plan satisfaction)
- How helpful is this application for you to find entertainment plans comparing with the other service mediator websites? (Helpfulness to users)
- How much Web technical knowledge is required for using this application? (Requirement of technical background)

The survey provides five choices for each question to measure the degree of the answer, such as “extremely helpful”, “very helpful”, “moderately helpful”, “slightly helpful” and “not at all helpful”. Those answers are mapped to the score of 0-5 means the most positive answer and 0 represents the most negative answer.

6.9.3 Results

Results for Evaluating the Performance of Service Discovery and Composition

We calculate the time consumption of service discovery & mashup depending on the number of service types. Figure 32 shows the result of service discovery & mashup experiment from one type (shopping) to five types (shopping, restaurant, museum, movie, park). We can see the time cost for those five situations fluctuate from 2 seconds to 3 seconds because we use the multi-thread to do service discovery. For

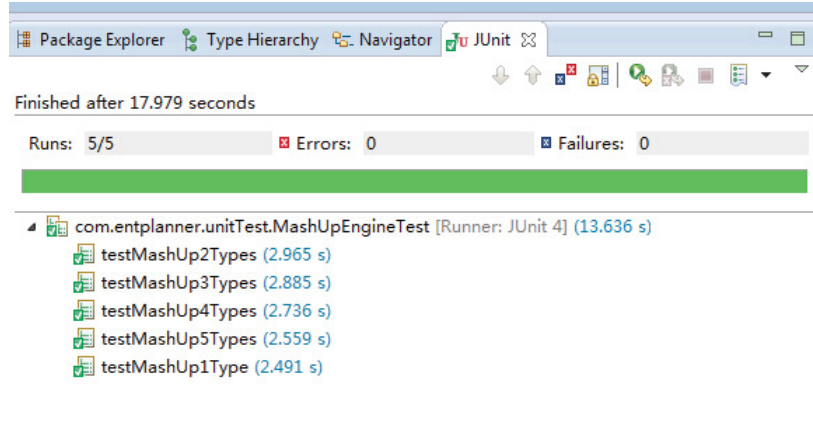


Figure 32: Results for Service Discovery

Table 14: Results for Service Composition

Number of service types	Number of combinations	Average time cost (s)
1	5	0.3818
2	10	0.825
3	10	1.3251
4	5	1.744
5	1	2.022

example, if we need to find five types of service, 5 parallel tasks will be created. The time cost of service discovery is related to only one of these five tasks which needs the maximum time.

Table 14 indicates the trend line of time consumption of service composition from 1 type to 5 types. The ideal approach should ensure the time cost for processing is acceptable and stable. As shown in Table 14, the time consumptions of service composition for the five test cases are increased from 0.4 seconds to 2 seconds with the rise of the types of services we compose. The time consumption of service composition is depended on the number of types of services we compose.

Result for Evaluating the User Experience of the Whole System

Table 15 lists the average value of each question in our survey. In Table 15, the first column shows the question number related to the questions in the survey. The last column indicates the average value of the scores provided by the subjects. The results

Table 15: Results of Satisfaction Evaluation

Question no.	Metrics	Average value
1	Satisfaction of response time	4.5
2	Success rate of generating plans	4.7
3	Plan satisfaction	4.1
4	Helpfulness to users	4.2
5	Requirement of technical background	0.5

show that most of subjects were satisfied with the response time of the process of generating plans. From the success rate of generating plans, we can see our approach can always provide users plans based on their requirements. The plans our process produced are acceptable for most of subjects and make it easy for subjects to build plans for entertainment. Moreover, our process is independent from a user's familiarity with the technical knowledge of Web services.

Chapter 7

Conclusion

This thesis starts with problem and motivation. Nowadays, Automated Service Composition as an approach related to Web services has drawn a lot of attention. Many researchers only concentrate on the electronic services, *i.e.*, automatic services that are provided by software systems. However, many real business services in our life do not attract a lot of attention such as restaurants, movie theatres and retail stores, which are non-electronic services. The scope of service computing should cover all kinds of services, including both electronic services and non-electronic services. In this thesis, we try to investigate the possibilities to do the non-electronic service discovery and composition. The service discovery and service composition are able to be done by employing context information such as location, identity, and time. In other words, context information could be used as a part of constraints for doing service discovery and service composition.

Currently, for a Web application, contextual information could be detected by HTML5 supported Web browsers or mobile Web browsers or provided by the user. In order to model the context information, we propose a model for context representation (Chapter 3). Our context model is built on the top of one ontology. A part of the ontology is domain independent, while the domain dependent ontology can also be extended easily. This model is able to handle both logic values and real values. We have an ability to represent the current situation of an entity or a person by adopting this model.

We developed a method to discover different types of non-electronic services over the Internet, which mainly include direction services, movie services and other business

services from several service search engines. Some of services are RESTful services, the others are HTML services. The types of non-electronic services can be added easily by using this method. In order to compose different types of services, we build a mashup mechanism to uniform the format of the data coming from the different types of services. This mashup mechanism can be regarded as a bridge between those original services and the composition function.

Based on the context model and service discovery mashup, we provide a planning based service composition approach to build a personal entertainment planner since this service composition problem can be seen as a planning problem with constraints [NFF⁺05] (Chapter 5). We use a backtracking search algorithm to solve this service composition problem (Section 5.2). The backtracking algorithm use Depth-First search to build its search tree which can includes all the possibilities of the problem. Moreover, the values of some constraints can only be calculated until we determine the prior state for the current state. In other word, we can do calculations during the period of building the tree. Additionally, one new feature was added into the service composition algorithm to make the solutions be diversified. In this thesis, we calculated the Levenshtein distance between two solutions in order to ensure the diversity. We also created a function to replan (doing service composition again based on a user's context if a user does not satisfy with the original solution).

Taking advantages of all the parts above, we designed a personal entertainment planner which is a Web based system. This planner is compatible with normal Web browsers and mobile Web browsers on mobile phone because it has two versions of UI (desktop and mobile). This system use three parts to implement context modeling, service discovery, service mashup and service composition. Those three parts are User View Application, Service Process Application and Service Discovery Application. We wrap the service composition applicaiton and the service discovery application as RESTful services. Those applications use HTTP protocol to communicate. All the results of this planner are stored in a database. The architecture of this system is easy for extension and maintenance in the future (Section 6.3).

In this thesis, we use the personal entertainment planner as a prototype application to demonstrate our procedure of context-aware non-electronic service discovery and composition. After that, we also create some tests for evaluating the performance and users' experience. Our goal is to check whether the time consumptions for

service discovery and composition are acceptable. From the results of tests, both time consumptions of service discovery and composition are acceptable. From the results of satisfaction evaluation, most of users are satisfied with the response time and success rate of generating plans. However, we still need to improve the quality of plans.

In the next step, we plan to add some new evaluation criteria to make solutions (plans) fulfill users' expectation closely, *e.g.*, using user's customized preference to evaluate solutions. In order to enhance the performance of service composition, we also need to make services' information more precise and expand the types of service.

Bibliography

- [AGS⁺93] Norman Adams, Rich Gold, Bill N Schilit, Michael M Tso, and Roy Want. An infrared network for mobile computers. In *Proceedings USENIX Symposium on Mobile & Location-independent Computing*, volume 10, 1993.
- [All83] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [API13] Yellow API. Yellow api. <http://www.yellowapi.com/>, 2013. Retrieved 2014-01-02.
- [BBH⁺10] Claudio Bettini, Oliver Brdiczka, Karen Henriksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161 – 180, 2010. Context Modelling, Reasoning and Management.
- [BC97] Bruno Bouzy and Tristan Cazenave. Using the object oriented paradigm to model context in computer go. In *Proceedings of the First International and Interdisciplinary Conference on Modeling and Using Context*, pages 279–289, 1997.
- [BDS08] Djamal Benslimane, Schahram Dustdar, and Amit Sheth. Services mashups. *The New Generation of Web Applications*, pages 13–15, 2008.
- [BEK⁺00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (soap) 1.1, 2000.

- [BHH04] Sven Buchholz, Thomas Hamann, and Gerald Hübsch. Comprehensive structured context profiles (cscp): Design and experiences. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, PERCOMW '04*, pages 43–, Washington, DC, USA, 2004. IEEE Computer Society.
- [BSR10] Roman Barták, Miguel A Salido, and Francesca Rossi. Constraint satisfaction techniques in planning and scheduling. *Journal of Intelligent Manufacturing*, 21(1):5–15, 2010.
- [CAH06] Daniela Barreiro Claro, Patrick Albers, and Jin-Kao Hao. Web services composition. In *Semantic Web Services, Processes and Applications*, pages 195–225. Springer, 2006.
- [CBJC11] Bachir Chihani, Emmanuel Bertin, Fabrice Jeanne, and Noel Crespi. Context-aware systems: a case study. In *Digital Information and Communication Technology and Its Applications*, pages 718–732. Springer, 2011.
- [CF03] Ekaterina Chtcherbina and Marquart Franz. Peer-to-peer coordination framework (p2pc): Enabler of mobile ad-hoc networking for medicine, business, and entertainment. In *Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet, LAquila, Italy*, pages 883–891, 2003.
- [Che76] Peter Pin-Shan Chen. The entity-relationship model toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.
- [CIJ⁺00] 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.
- [CK⁺00] Guanling Chen, David Kotz, et al. A survey of context-aware mobile computing research. Technical report, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

- [CMD99] Keith Cheverst, Keith Mitchell, and Nigel Davies. Design of an object model for a context sensitive tourist guide. *Computers & Graphics*, 23(6):883–891, 1999.
- [DB03] Jos De Bruijn. Using ontologies-enabling knowledge sharing and reuse on the semantic web. 2003.
- [Dey01] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, January 2001.
- [DF99] Rina Dechter and Daniel Frost. Backtracking algorithms for constraint satisfaction problems. Technical report, Citeseer, 1999.
- [DHM⁺04] 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.
- [DPAM07] Luciano A Digiampietri, José J Pérez-Alcázar, and Claudia Bauzer Medeiros. Ai planning in web services composition: a review of current approaches and a new solution. *SBC*, 2007:983–992, 2007.
- [EFKS10] Sergei Evdokimov, Benjamin Fabian, Steffen Kunz, and Nina Schoenemann. Comparison of discovery service architectures for the internet of things. In *Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC), 2010 IEEE International Conference on*, pages 237–244. IEEE, 2010.
- [Exp13] Expedia. Expedia. <http://www.expedia.ca>, 2013. Retrieved 2014-01-02.
- [Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.
- [Fie02] Richard N Fielding, Roy T. and Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, pages 115–150, 2002.
- [Fou13] Foursquare. Foursquare api. <https://developer.foursquare.com/>, 2013. Retrieved 2014-01-02.

- [GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Elsevier, 2004.
- [Goo13a] Google. Google maps api. <https://developers.google.com/maps/>, 2013. Retrieved 2014-01-02.
- [Goo13b] Google. Google places api. <https://developers.google.com/places/>, 2013. Retrieved 2014-01-02.
- [HH12] David Harris and Sarah Harris. *Digital design and computer architecture*. Elsevier, 2012.
- [HIR02] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Pervasive Computing*, pages 167–180. Springer, 2002.
- [HIR03] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Generating context management infrastructure from high-level context models. In *In 4th International Conference on Mobile Data Management (MDM)-Industrial Track*. Citeseer, 2003.
- [HTD90] James A Hendler, Austin Tate, and Mark Drummond. Ai planning: Systems and techniques. *AI magazine*, 11(2):61, 1990.
- [IBM08] IBM. Restful web services: The basics. <http://www.ibm.com/developerworks/library/ws-restful/>, 2008.
- [IEE03] IEEE Technical Committee. Services computing. <http://tab.computer.org/tcsc/>, 2003. Retrieved 2014-03-15.
- [KKC⁺01] Lalana Kagal, Vlad Korolev, Harry Chen, Anupam Joshi, and Timothy Finin. Centaurus: A framework for intelligent services in a mobile environment. In *Distributed Computing Systems Workshop, 2001 International Conference on*, pages 195–201. IEEE, 2001.
- [KL03] Rania Khalaf and Frank Leymann. On web services aggregation. In *Technologies for E-Services*, pages 1–13. Springer, 2003.

- [KS07] Reto Krummenacher and Thomas Strang. Ontology-based context modeling. In *Proceedings Third Workshop on Context-Aware Proactive Systems (CAPS 2007)(June 2007)*, page 22, 2007.
- [LB03] Adriana Lopez and Fahiem Bacchus. Generalizing graphplan by formulating planning as a csp. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, IJCAI'03*, pages 954–960, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.
- [LMB07] Anany Levitin, Soumen Mukherjee, and Arup Kumar Bhattacharjee. *Introduction to the design & analysis of algorithms*, volume 2. Pearson Addison-Wesley, 2007.
- [LT07] Iain Little and Sylvie Thiebaut. Probabilistic planning vs. replanning. In *ICAPS Workshop on IPC: Past, Present and Future*, 2007.
- [Lue00] Christopher Lueg. Context-awareness and context-transparency as orthogonal concepts in hci. Technical report, Technical Report IFI-AI, 2000.
- [MB97] John McCarthy and Sasa Buvac. Formalizing context (expanded notes). 1997.
- [MBE03] Brahim Medjahed, Athman Bouguettaya, and Ahmed K Elmagarmid. Composing web services on the semantic web. *The VLDB Journal*, 12(4):333–351, 2003.
- [MBH⁺04] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srinu Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, et al. Owl-s: Semantic markup for web services. *W3C member submission*, 22:2007–04, 2004.
- [MC12] Debajyoti Mukhopadhyay and Archana Chougule. A survey on web service discovery approaches. In *Advances in Computer Science, Engineering & Applications*, pages 1001–1012. Springer, 2012.
- [McC93] John McCarthy. Notes on formalizing context. 1993.

- [MGH⁺98] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. 1998.
- [MJ03] Teddy Mantoro and Chris Johnson. Location history in a low-cost context awareness environment. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003-Volume 21*, pages 153–158. Australian Computer Society, Inc., 2003.
- [MSF⁺00] Stuart Madnick, Michael Siegel, Mary Alice Frontini, Saraubh Khemka, Steven Chan, and Howard Pan. Surviving and thriving in the new world of web aggregators. In *unpublished paper presented at the Society for Information Management Workshop, Brisbane, Australia, December, 2000*.
- [NFF⁺05] Alexander Nareyek, Eugene C Freuder, Robert Fourer, Enrico Giunchiglia, Robert P Goldman, Henry Kautz, Jussi Rintanen, and Austin Tate. Constraints and ai planning. *Intelligent Systems, IEEE*, 20(2):62–72, 2005.
- [Nie00] Jakob Nielsen. Why you only need to test with 5 users, 2000.
- [ÖA97] Pinar Öztürk and Agnar Aamodt. Towards a model of context for case-based diagnostic problem solving. In *Context-97; Proceedings of the interdisciplinary conference on modeling and using context*, pages 198–208. Citeseer, 1997.
- [OAS07] OASIS. Uddi version 2.04 api specification. <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm>, 2007.
- [Ora06] Oracle. Restful web services. <http://www.oracle.com/technetwork/articles/javase/index-137171.html>, 2006.
- [Pee09] Sunilkumar Peenikal. Mashups and the enterprise. *Strategic white paper*, 2009.
- [RL10] Wenge Rong and Kecheng Liu. A survey of context aware web service discovery: from user’s perspective. In *Service Oriented System Engineering*

- (SOSE), *2010 Fifth IEEE International Symposium on*, pages 15–22. IEEE, 2010.
- [RNC⁺95] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Englewood Cliffs, 1995.
- [RS05] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition*, pages 43–54. Springer, 2005.
- [Rya99] Nick Ryan. Contextml: Exchanging contextual information between a mobile client and the fieldnote server. 1999, 1999.
- [SAG⁺93] Bill N Schilit, Norman Adams, Rich Gold, Michael M Tso, and Roy Want. The parctab mobile computing system. In *Workstation Operating Systems, 1993. Proceedings., Fourth Workshop on*, pages 34–39. IEEE, 1993.
- [SAW94] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 85–90. IEEE, 1994.
- [See12] Seekda. Web services search engine. webservices.seekda.com, 2012.
- [SFV⁺95] Thomas Schiex, H elene Fargier, Gerard Verfaillie, et al. Valued constraint satisfaction problems: Hard and easy problems. *IJCAI (1)*, 95:631–639, 1995.
- [SGCB00] Hans Schuster, Dimitrios Georgakopoulos, Andrzej Cichocki, and Donald Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In *Advanced Information Systems Engineering*, pages 247–263. Springer, 2000.
- [SLP04] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop Proceedings*, 2004.
- [SMA07] Vincenzo Suraci, Silvano Mignanti, and Anna Aiuto. Context-aware semantic service discovery. In *Mobile and Wireless Communications Summit, 2007. 16th IST*, pages 1–5. IEEE, 2007.

- [SMLP02] Michael Samulowitz, Florian Michahelles, and Claudia Linnhoff-Popien. Capeus: An architecture for context-aware selection and execution of services. In *New developments in distributed applications and interoperable systems*, pages 23–39. Springer, 2002.
- [Sri13] Vivek Srikumar. Soft constraints in integer linear programs. 2013.
- [Str03] Thomas Strang. *Service Interoperability in Ubiquitous Computing Environments*. PhD thesis, Ludwig-Maximilians-University Munich, October 2003.
- [Tri13] Tripadvisor. Tripadvisor. <http://www.tripadvisor.ca>, 2013. Retrieved 2014-01-02.
- [W3C04] W3C. Web services glossary. <http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/>. Retrieved 2011-04-22, 2004.
- [W3C07a] W3C. Soap version 1.2 part 1: Messaging framework (second edition). <http://www.w3.org/TR/soap12-part1/#intro>. Retrieved 2011-06-30, 2007.
- [W3C07b] W3C. Web services description language (wsdl) version 2.0. <http://www.w3.org/TR/wsdl20/>. Retrieved 2011-06-30, 2007.
- [W3C14a] W3C. Composite capabilities/preferences profile. <http://www.w3.org/Mobile/CCPP/>, 2014.
- [W3C14b] W3C. Html 5.1. <http://www.w3.org/TR/html51/>, 2014.
- [W3C14c] W3C. Resource description framework. <http://www.w3.org/RDF/>, 2014.
- [WAP14] WAPFORUM. User agent profile (uaprof). <http://www.wapforum.org>, 2014.
- [Wik13] Wikipedia. Allen’s interval algebra. http://en.wikipedia.org/wiki/Allen's_interval_algebra, 2013. Retrieved 2014-02-02.
- [Wik14a] Wikipedia. Automated planning and scheduling. http://en.wikipedia.org/wiki/Automated_planning_and_scheduling, 2014. Retrieved 2014-02-02.

- [Wik14b] Wikipedia. Levenshtein distance. http://en.wikipedia.org/wiki/Levenshtein_distance, 2014.
- [Wik14c] Wikipedia. W3c geolocation api. http://en.wikipedia.org/wiki/W3C_Geolocation_API, 2014.
- [Wik14d] Wikipedia. Web application. http://en.wikipedia.org/wiki/Web_application, 2014.
- [Wik14e] Wikipedia. Web services description language. http://en.wikipedia.org/wiki/Web_Services_Description_Language, 2014.
- [WZGP04] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 18–22. Ieee, 2004.
- [Yel13] Yelp. Yelp v2.0 api. <http://www.yelp.com/developers/documentation>, 2013. Retrieved 2014-01-02.
- [YZ08] Yuhong Yan and Xianrong Zheng. A planning graph based algorithm for semantic web service composition. *CEC/EEE*, 2008:339–342, 2008.
- [Zha96] W. Zhang. Branch-and-Bound Search Algorithms and Their Computational Complexity. Technical Report ISI/RR-96-443, Univ. of southern California/ Information Sciences Inst., 1996.
- [ZSM01] Hongwei Zhu, Michael D Siegel, and Stuart E Madnick. Information aggregation—a value-added e-service. In *Proc. of the International Conference on Technology, Policy, and Innovation: Critical Infrastructures*, 2001.
- [ZY08] Xianrong Zheng and Yuhong Yan. An efficient syntactic web service composition algorithm based on the planning graph model. In *Web Services, 2008. ICWS'08. IEEE International Conference on*, pages 691–699. IEEE, 2008.

- [ZZL10] Yilei Zhang, Zibin Zheng, and Michael R Lyu. Wsexpress: A qos-aware search engine for web services. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 91–98. IEEE, 2010.