CONTEXT-AWARE SERVICE DISCOVERY AND SERVICE COMPOSITION OVER SMART PHONES

Jia Ning Wang

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

March 2013

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Jia Ning Wang

Entitled: Context-aware service discovery and service composition over smart phones

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

Master of Computer Science

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

Signed by the final examining committee:

Nikolaos Tsantalis	Chair
Sabine Bergler	Examiner
Thiruvengadam Radhakrishnan	Examiner
Yuhong Yan	Supervisor

Approved by Peter Grogono
Chair of Department or Graduate Program Director

Robin Drew
Dean of Faculty

Date April 14, 2013

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By:	**Jia Ning Wang**
Entitled:	**Context-aware service discovery and service composition over smart phones**

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

Master of Computer Science

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

Signed by the final examining commitee:

______________________ Chair
______________________ Examiner
______________________ Examiner
______________________ Examiner
______________________ Supervisor

Approved ______________________
Chair of Department or Graduate Program Director

__________ 20 _____ ______________________
Rama Bhat, Ph.D.,ing., FEIC, FCSME, FASME, Interim Dean
Faculty of Engineering and Computer Science

Abstract

Context-aware service discovery and service composition over smart phones

Jia Ning Wang

One of the advantages of smart phones is the ubiquity of sensing and computing power. The smart phone consists of a mobile computing platform. The Web browsers installed on the platform make it possible to surf the Internet through mobile broadband and Wi-Fi. An important aspect of smart phones is that they have application programming interfaces, which is able to take advantage of third-party applications. Different from any desktop applications, the smart phone applications could be highly adaptive to contexts, *i.e.* according to context information, *e.g.* location, identity, and time, the applications are tuned to satisfy particular requirements in the contexts. On the other sense, service composition is a way to plan a business process to fulfill business goals that cannot be achieved by individual business services. Service composition can be modeled as a AI planning problem. Based on the initial context and the goal context, planning-based service composition launches a goal-oriented composition procedure to generate a plan. Service composition over smart phones can be context-awareness. In this thesis, we want to investigate context based service discovery and service composition over smart phones. We propose a constraint-based context model. We include non-electronic services into service composition, which extends the scope of services considered in existing service composition research. Moreover, our composition algorithm suits mobile computation power because the service composition can adjust to the computation power of mobile phones easily. As a motivating example, we build an entertainment planner over an Android phone.

Acknowledgments

I would like to express my gratitude and thanks to my supervisor Dr. Yuhong Yan for all her help, guidance and supervision during the production of this thesis. Without her brilliant ideas this thesis would not have been possible.

I also would like to take this opportunity to thank my friends who give me lots of support and help, especially, Min Chen. Thank you for all the fruitful discussion and encouragement.

Finally, the deepest gratitude goes to my family for all your caring, support and love. Thank you for the comforts from the beginning to the end.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem and Motivation

Nowadays, the smart phone becomes the most common type of mobile computing device. The fast development of software and hardware makes a smart phone have the functionality of a small computer. In addition to making telephone calls, a smart phone has many other functions, such as the ability of connecting to the Internet through mobile broadband and Wi-Fi, GPS navigation, video camera, digital camera, and media players. Moreover, smart phones consist of mobile operating systems, and have application programming interfaces (API). This means that smart phones can take advantage of applications, which are fully integrated with its hardware and operating system.

Smart phones have location awareness. Nearly all smart phones are equipped with a global position system (GPS). The GPS in a smart phone can determine the device's current location in real time. This allows the opportunity of designing location-aware applications. Contextual-aware computing has been explored as a way of using situational information to achieve goals of providing people with applications.

Since the smart phones have access to different type of services over the Internet, it is possible to design an application through service composition to fulfill new functions. Service composition is to create a business process by connecting multiple services so as to fulfill some business goals that individual services cannot achieve. Normally, service composition assumes that data sources provided by services are on a consistent basis. However, in the real world, different types of services may return the data in different format. This adds to the complexity of the problem.

A service composition procedure starts from service discovery and service selection. Then the composition algorithm automatically generates a target business process (or processes) with the provided service sets. Business services studied by service computing should not be limited to the electronically implemented ones. For example, the UDDI standard covers all kinds of business services including retailers and restaurants. However, most of the existing research on service composition tends to focus on composing the electronic services.

1.2 Contribution

In this paper, we study context-aware service composition problem over smart phone devices. We build an entertainment planner on an Android phone as an example to demonstrate the procedure of context-aware service discovery and service composition over smart phones. Our main contributions include:

1. We propose a constraint-based context model. In order to ease knowledge sharing and reuse knowledge for future extension, we firstly build a domain related ontology for context. Then, a constraint-based context model is build on the top of the ontology. Moreover, our constraint-based context model are able to deal with both propositions and real values. Using our context model, we can express both the current context and the business goals. We also realize context-awareness in service composition based on the constraint-based context model.

2. We extend the services in service composition into non-electronic services such that service composition can be performed on all kinds of services. Normally, people limit the services to electronic services in existing research where the inputs of a service are the output of another service. But the scope of service computing should include non-electronic services as well, for example, all businesses in a standard like North American Industry Classification System (NAICS). In this thesis, we present a way to discover and compose not only the electronic services but also the non-electronic services. The preconditions and the post conditions of the services are modeled as contexts and the constraints determine the execution order of the services in a composite business process.

3. We develop a service composition algorithm based on our service discovery mashup suitable for a smart phone. Service composition employs the context information obtained either from user's inputs or from the smart phones system. The capabilities of connecting to the Internet, detecting current position through GPS (Global Positioning System), and hardware improvement make it possible to perform service composition over smart phones. Moreover, our service composition is implemented on smart phones platform such that service composition can quickly response to user's request.

1.3 Organization of the Thesis

The structure of this thesis is as follows: the first chapter starts with the introduction of the problem, the motivation and the contributions. The second chapter gives an overview of Web Services, service mashup and composition, and context representation and computing in particular. Later in this chapter, we introduce a motivating example. The third chapter presents the related definitions in constraint model for context representation. We discuss composed service discovery and mashup in the fourth chapter. In this chapter, we elaborate three different types of Web services in our planner and how to mashup them. The fifth chapter covers the details related to algorithm of Web service composition. The sixth chapter describes the implementation of the entertainment planner. The last chapter concludes the thesis by summarizing the contributions and pointing out possible future work.

Chapter 2

Fundamentals

2.1 Web Services

Given the definition by the World Wide Web Consortium (W3C) [W3C04b] ,

> 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.

Web Services consist of computational elements, which interact with each other yet are independent [Pie11]. These elements can perform a wide variety of tasks, which range from simple request to complex processes, which are often used in business. Most of these elements communicate with each other through messages using Extensible Markup Language (XML). These messages follow the Simple Object Access Protocol (SOAP). These service elements are platform independent, self-contained, and self-describing. They can be accessed using standard Internet protocols [CW10].

The basic architecture of Web services is shown in Figure 1. There are three roles in the Web service architecture, service requesters, service providers, and service brokers. Service providers provide the actual services. They implement Web services and serve them online by publishing their associated WSDL document, which contain the way to invoke those services they provide, to a service broker using UDDI specification. Service requesters locate the services from a service broker and then invoke the service from service requesters via SOAP protocol with the address specified in the WSDL document.

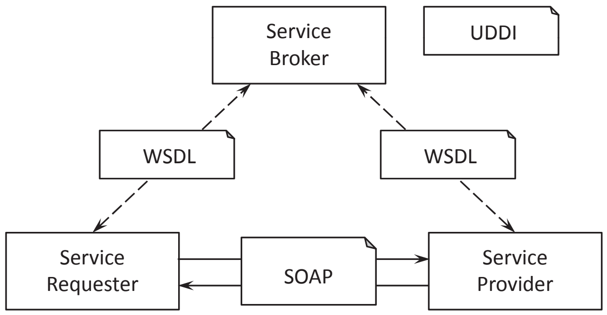


Figure 1: Web Service Architecture [Wik12c]

Web Services are now an important technology which allows for automated interactions between heterogeneous and distributed applications [BKH11]. There are underlying technologies, which make these Web service elements possible. These elements are XML, SOAP, WSDL, and Universal Description Discovery and Integration (UDDI).

2.1.1 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is used to exchange structured information for implementation of specific Web services within computer networks [TAY10]. The protocol makes use of XML and frequently uses other application layer protocols. These can include Simple Mail Transfer Protocol (SMTP) and Hypertext Transfer Protocol (HTTP) [Pie11].

The SOAP is frequently used as part of the Web services protocol stack [Pie11]. In this case, SOAP will be the foundation layer upon which the stack is built. This means SOAP is providing the basic framework upon which messaging between services can occur. The protocol based on XML has three parts. These parts include the envelope, encoding rules, and representation of procedure responses and calls. An advantage of SOAP is that it can be used with a number of different transport protocols including Job of Message Service (JMS), Transmission Control Protocol (TCP), SMTP, and HTTP [Pie11].

Either HTTP or SMTP can be used as the application layer protocol which is used to transport SOAP [Pie11]. However, HTTP is more widely used due to its common use as part of the infrastructure involved with the Internet. One advantage of HTTP is that it integrates well with most network firewalls [TAY10].

Another advantage of SOAP is that it can be used with Hypertext Transfer Protocol Secure (HTTPS) [Pie11]. This is a type of communication protocol, which is widely used with computer networks that interact over the Internet. It is actually not a standalone protocol. It consists of HTTP, which is layered on the Transport Layer Security (TLS) or Secure Sockets Layer (SSL). These are both cryptographic protocols, which allow for secure communication using the Internet. Both protocols encrypt a segment of the network connection using a form of asymmetric cryptography. In order for privacy to be achieved, symmetric encryption is used. Message integrity requires the use of message authentication codes [Pie11].

SOAP is not without its disadvantages [TAY10]. The XML format can be significantly slower than middle-ware technologies, which include the Common Object Request Broker Architecture (CORBA). With regard to small messages, this is often not a problem. However, the CORBA outperforms the SOAP when longer in complex messages are being sent. Part of this problem has been alleviated with the Message Transmission Optimization Mechanism (MTOM) which allows for the efficient transfer of binary data between services [Pie11].

2.1.2 Web Services Description Language (WSDL)

Web Services need to employ a structured method (XML) to describe the invocation and communication between them. The Web Services Description Language (WSDL) specification provides an XML format for documents for this purpose [Wik12b]. The WSDL provides an official description for Web Services in a machine-readable way. It is an XML-based language that is used for describing Web Services and how the services can be called, what parameters it expects, and what data structures it returns. The WSDL defines Web Services as collections of “ports” (WSDL 1.1) or “endpoints” (WSDL 2.0) (see Figure 2). Each “port” is associated a network address with a reusable binding. And the binding section defines the “operations” (WSDL 1.1) or “interfaces” (WSDL 2.0) to be related with supported operations of one Web service. Each “operation” is a SOAP action called by service requesters, which is like a method or function call in a traditional programming language. The format of “input message” and “output message” in an “operation” is defined at “message” sections and the data types used are defined in the “types” sections.

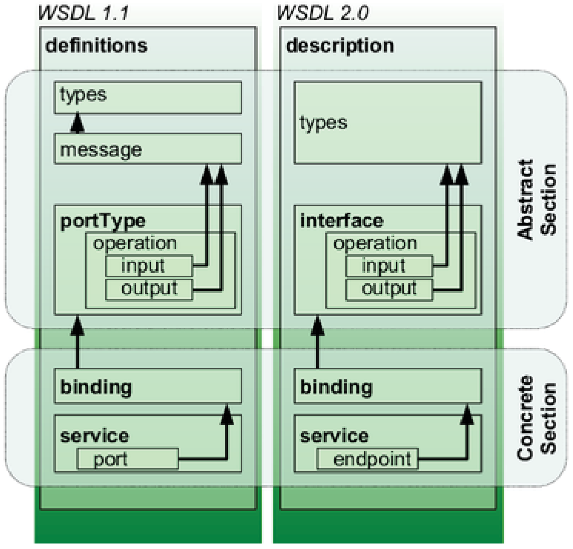


Figure 2: Layered Structure of WSDL [Wik12b]

2.1.3 Universal Description Discovery and Integration (UDDI)

Universal Description Discovery and Integration (UDDI) is an Extensible Markup Language (XML) based registry, which is used by businesses throughout the world in order to list their organization on the Internet. It also serves to locate and register Web service applications [Sob10]. The UDDI is an industry initiative which is sponsored by the Organization for the Advancement of Structured Information Standards (OASIS). This organization hopes that UDDI will help businesses discover each other through published service listings. The UDDI can be thought of as an industry standard which allows interactivity for Web service applications over the Internet. Originally, the UDDI was proposed to be a standard core Web service which could be interrogated by Simple Object Access Protocol (SOAP). This would allow access to the Web Services Description Language (WSDL) based documents. These documents provide a description of the message formats and protocol bindings which are required for interaction to occur with Web services [Zel10].

The UDDI registration of a business consists of three components [Sob10]. These components are the green, yellow, and white pages. The green pages provide technical information regarding services, which are exposed by the business through the business being registered. The yellow pages are categorizations, which are based on standardized taxonomies. The white pages contain specific known identifiers, contact information, and the business address [Zel10].

The white pages usually contain contact information for an organization, including items such as its business address as well as a telephone number. There may also be ratings of credit worthiness and other important information [Sob10].

The yellow pages are the business classification according to a standardized taxonomic system [Zel10]. Standards were set by the United Nations Standard Products and Services Code (UNSPSC), the North American Industry Classification System (NAICS), and the Standard Industrial Classification (SIC).

The green pages provide information for accessing certain Web services [Zel10]. For example, the Internet address of the service as well as its parameters may be listed in these pages. There may be information on interfaces and specifications of the service. There may also be information regarding e-mail addresses. Some Web services have multiple bindings, in which case, there will be several green pages. Each page will correspond to a single binding [Zel10].

2.1.4 Representational State Transfer (RESTful)

A rival that challenged SOAP was the RESTful style of Web services in 2000 at the University of California [IBM08]. It comprises of simple communication over HTTP with the help of old XML versions. This technology was based upon the concept of "Representative State Transfer" put forward by Roy Fielding in his thesis [Fie00]. REST attempts to describe architectures that use HTTP or similar protocols by constraining the interface to a set of well-known, standard operations (like GET, POST, PUT, DELETE for HTTP). Here, the focus is on interacting with stateless resources, rather than messages or operations. Clean URLs are tightly associated with the REST concept.

REST is based upon the following four basic design principles:

1. Explicit use of HTTP methods,
2. Being stateless,
3. Exposure of URIs similar to a directory structure, and
4. Transfer of XML or JavaScript Object Notation (JSON), or both.

REST can communicate between various clients written in different languages and focuses on the system's resources. Its ease of implementation and use has made it a more predominant Web service design model. In many places it has displaced SOAP and WSDL–based interface designs [IBM08]. Besides, an architecture based on REST can use WSDL to describe SOAP messaging over HTTP, can be implemented as an abstraction purely on top of SOAP (*e.g.* WS-Transfer), or can be created without using SOAP at all.

The recently developed type of service abstraction for Web service composition is the Representational State Transfer (RESTful). This is the language frequently used for Web service composition. The RESTful approach allows for interaction primitives to be directly mapped and published to Web applications, which already exist. This means that the Hypertext Markup Language (HTML) pages are replaced with data payloads using XML [BKH11].

2.1.5 SOAP vs. REST

Both these technologies have their advantages and disadvantages. For example, SOAP is designed to handle distributed computing environments, has better support from other standards like WSDL and WS-*, has built-in error handling and is extensible. Its disadvantages include conceptual difficulties, being more verbose, difficulty in developing and the tools required in development. However, it becomes a better choice when a formal contract is required to elaborate the interface offered by the Web service. Apart from that, it is also suitable for situations like the presence and handling of complex non-functional requirements and the need for the architecture to handle asynchronous processing and calls [Ora06].

On the other hand, REST also has its pros and cons. Its advantages include its simplicity to develop; less reliance on tools and, hence, less learning curve; its concise and its closeness to Web in design and philosophy. Its drawbacks are that its point-to-point communication model is not usable when it faces distributed computing environments; does not have standard support for security, policy, reliable messaging, etc.; and that it is tied to the HTTP transport model.

REST is recommended in the following situations [Ora06]:

- Completely stateless Web services;

- The presence of control over caching infrastructure for better performance;
- The producers and the consumers of the service have mutual understanding of the context and content;
- Limited bandwidth – REST is more useful in such environments like PDAs and mobile phones;
- Where integration of Web service delivery and existing Web sites is required – AJAX and DWR can be used to consume the services in the Web applications.

Apart from the above, Java API for XML Web Services (JAX-WS) is in full support for building and deploying Web services.

With their own pros and cons, both SOAP and REST have attractions for the designers and developers. It is their responsibility to evaluate their needs and make the right decision.

2.1.6 Data as a Service(DaaS)

Data as a service (DaaS) is similar to software as a service [CW10]. The DaaS has data as a product which can be provided when needed by the customer. When this information is provided over the Internet, it can be sent to the consumer in any location, which has a World Wide Web connection. This means the geographic location of the customer is irrelevant. It also does not matter what type of platform the data is being stored on. The popularity of service oriented architecture (SOA) has solved this problem as well [BKH11].

When DaaS was introduced, it was used primarily in Web mashups [CW10]. These consist of aggregation, visualization, and combination. They allow the data which already exist to be more useful. Many mashups are posted online or are client applications. However, DaaS is now being widely used as a commercial service. Furthermore, there are a number of international organizations such as the United Nations which make use of DaaS repositories for preparing reports and setting policies [CW10].

Until recently, the majority of organizations made use of data which they had stored in a repository which was self-contained [CW10]. This data could then be accessed through software, which would allow people to understand the data and give it meaning. This means that the software and data were generally bundled in a single package. This was then sold to the consumer as one product. There was a steady proliferation of the bundled software and data packages. This means there are an increasing number of interactions between these packages that required the creation of an interface layer. This became known as Enterprise Application Integration (EAI) [BKH11].

The EAI layer requires consumers to purchase additional middle-ware [CW10]. This means many organizations must spend substantial money in order to ensure that their hardware, software, and middle-ware are compatible. The software updates and maintenance costs can be high. This situation led to the formation of DaaS as an attractive option for many organizations and consumers. The use of DaaS allows for separation between the cost of the data and the software platform being used [CW10].

2.2 Context Representation and Computing

2.2.1 Context Modeling Approaches

Since the 90's, there has been an increase in research with regards to the use of context awareness as the method of developing advanced computing applications that are capable of working autonomously without much interference or control from the user [BBH$^+$10]. There are many benefits of formal context aware information modeling. Firstly, due to the complexity of these applications (context-aware), there is need of using adequate software engineering techniques, the main aim is to develop applications that are evolvable [BBH$^+$10]. Thus the applications designs should be independent from the definition and evolution of context information often subjected to change. High quality context information modeling reduces the complexity of these applications and enhances their ability to evolve and being maintainable. In addition, the cost of maintaining, gathering and evaluating context information is high thus, the sharing and re-use of context information between these applications that are context aware should be put in place at the beginning [BBH$^+$10]. The availability of well and professionally designed context information models enhances the development and deployment of more future applications. Further more, for consistency checking, there is need for a context data to be formally represented within a model [BBH$^+$10]. This also facilitates sound reasoning carried on context data. There are several context modeling approaches [Sch95, SAW94, Pas98, SBG99, CK00, Dey01, SLPF03], but we will look at the most relevant ones in a summarized manner.

In [Dey01], context is defined as any information that can be used to characterize the situation of an entity. The first step towards a common understanding of context are certain types of context, that are more important than others, with respect to *location, identity, activity* (also called *environment*) and *time*. The research on the use of context-awareness as a technique is to develop pervasive computing applications that are flexible and adaptable [BBH+10]. Context modeling and computing provides a uniform way to represent contexts and use contextual information.

In this thesis, available information obtained from smart phone devices is the context. In the existing work, context modeling approaches can be classified by the scheme of data structures [SLP04].

- Key-Value Models

  Key-value models use a set of attributes and their values such that each attribute and its value is a key-value pair to model context information.

  [SAW94] is a very early research work to use key-value models, which defines context-aware computing and 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 can examine and react to the changing context. [SAW94] thinks that context includes not only the user's location and time but also the located-objects. There are three types of located-objects. The first kind is the input and output devices nearby including printers, displays, and video cameras. The second kind is the set of objects that you are interacting with, such as people in the same room to whom you are talking. The third kind is the set of places nearby including restaurants, night clubs, and gas stations. An example of these attributes of the context is shown in Table 1.

  Table 1: Attributes and their values

Attribute	Value
Date and time	after April 13 between 10 and 12 noon
Location	in room 35-2200

  [SMLP01] also uses key-value models to represent the context which proposes a Context-Aware Packets Enabling Ubiquitous Service (CAPEUS) framework to realize context-aware selection and execution of services. In the framework, a mobile device acts as an access point for expressing service needs of the changing environment. CAPs (Context-Aware Packets) is designed to express service needs on a high abstraction level. CAPs consists of three parts: context constraints, scripting, and data. The context constraints include *abstract entities*, *relations*, and *events*. Figure 3 shows a printer entity. The attributes of the printer are a list of key-value pairs. A relation describes the dependencies of entities. One simple relation, *inRoom*, states that entities of this relation have to be in the same room. An event describes a trigger to execute a CAP initiated service call.

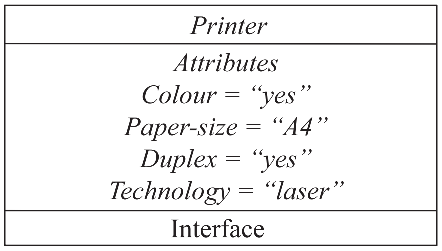


Figure 3: Printer Entity [SMLP01]

- Markup Schema Models

  Markup schema models use a variety of hierarchical data structures expressed with markup languages including XML. One example of this approach is to use CC/PP (Composition Capabilities / Preference Profiles) in context-aware systems [IRRH03]. CC/PP is an XML-based format standard developed by the W3C to describe the context information related to device capabilities and user preferences. When a device sends a request through the HTTP protocol, the request is a CC/PP profile. CC/PP is based on the Resource Description Framework (RDF), which consists of a set of statements. The context information to be captured in pervasive computing includes location information, user computing devices, computer application, operation system, user requirements and preferences, and computing application's requirement. For example, location information can be described as a Location profile as shown as below [IRRH03]:

```
[LocationProfile
   [PhysicalLocation [Country, State, City, Suburb]]
   [LogicalLocation [IPAddress]]
   [GeodeticLocation [Longitude, Latitude, Altitude]]
   [Orientation [Heading, Pitch]]
```

```
    [Modifications [VerticalError, HorizontalError, HeadingError, PitchError]]
]
```

LocationProfile is composed of several components: PhysicalLocation, LogicalLocation, GeodeticLocation, Orientation, and Modifications. Each component is described by several attributes. The relationships can also be represented as a profile as shown as below:

```
[UserDeviceProfile
    [User [URI]]
    [PermittedDevices [Bag of URI]]
]
```

The profile shows that the user at the given URI is permitted to use the devices identified by the RDF bag of URI. The "depends" relation can be described in the XML fragment as shown as below:

```
<ccpp:Attribute rdf:ID=‘‘bandwidth’’>
    <rdf:range rdf:resource=‘#QoS’>
    <constraints:depends rdf:resource=‘#batteryPower’>
</ccpp:Attribute>
```

Other context modeling approaches in this category are either limited to a small set of contextual aspects or the appropriateness of these approaches remain unknown [Hal01, Rya99, BBC97].

- Graphical Models

  Graphical models use Unified Modeling Language (UML) to represent the context in graphical structures.

  A representative work of graphical models is proposed in [HIR02]. It firstly introduces a case study of context-aware communication. In this case study, Bob and Alice plan to arrange a meeting through their communication agents. Therefore, communication agents will mediate the communication between Bob and Alice and give suggestions to Bob and Alice. The agents in fact collect all the context information about the participants and their communication devices and channels. In order to model the context information, [HIR02] uses both the Entity Relationship model and the class diagrams of UML to model the scenario of the case study as shown in Figure 4. There are three entity types: *Person*, *Device* and *Channel*. Each entity type has a number of attributes, *e.g.* *Person* has the attributes of *Activity*, *Name*, *Location Coordinates*. The associations between the entities and their attributes also capture the associations between the entities. For example, both *Person* and *Device* connect to the attribute of *Location Coordinates*. A dependency is a relation between associations, which shows the dependency of one association on another one. Figure 5 is the context model showing the derivation dependencies. For example, the dependency a_1 *dependsOn* a_2 is modeled as a directed arc leading from a_1 to a_2.

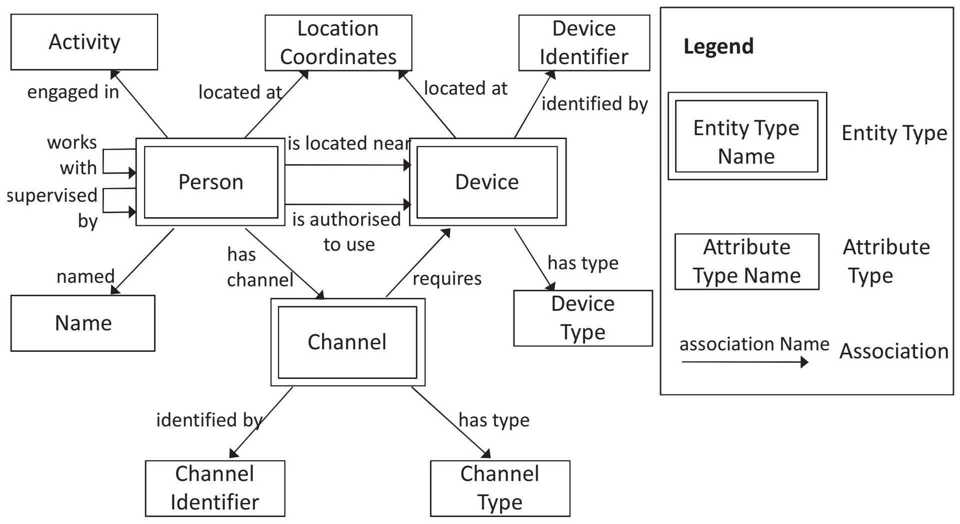


Figure 4: Modeling the scenario [HIR02]

There are also other research work using graphical models. Henrichsen et al. [HIR03] design graphics oriented context model, which is an extension of the Object-Role Modeling (ORM) approach [Hal01].

In all, graphical model is particularly appropriate for deriving an ER-model and useful to structure a relational database.

- Object Oriented Models

Object oriented context models take the benefit of any object oriented approach, namely the encapsulation and re-usability, for the ease of describing the dynamics of the context. A typical context-sensitive application using an

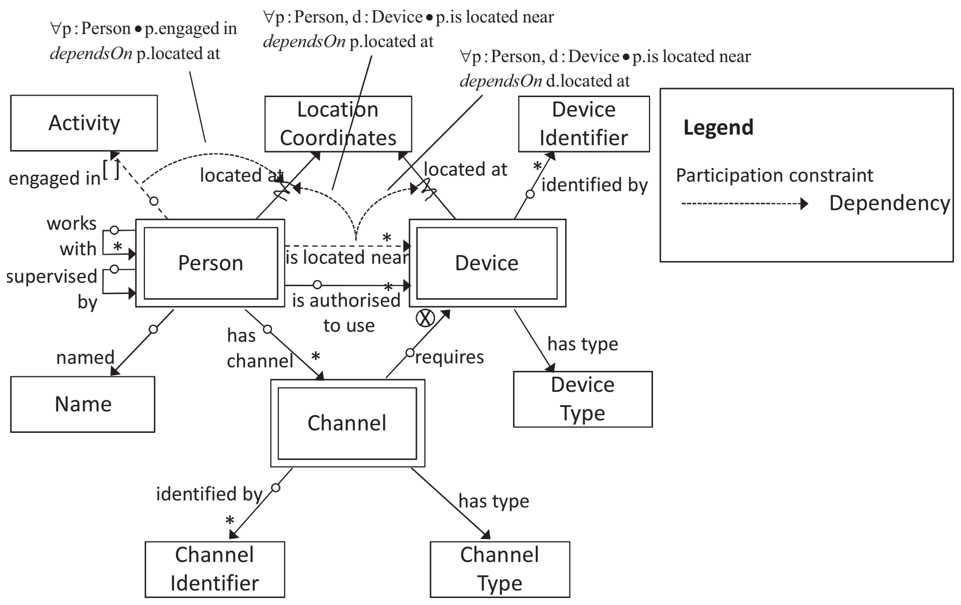


Figure 5: Context model showing the derivation dependencies [HIR02]

object model is the GUIDE project [CMD99] developed over pen-based tablet PCs and PDAs. The objective of the GUIDE project is to provide city visitors with recommended tours, a structured guide of the city or specific pieces of information along the way. There are two types of context-sensitive information modeled in the project, namely personal information and environmental information. Personal information includes the visitor's interests, the visitor's current location, the time visitors plan to spend on their visit, their budget and any refreshment preferences visitors might have. Environmental context includes the time of the day, the weather, and the current state of the city's transport system. The GUIDE object models are presented in Figure 6. A visitor sends his request through the local Web browser. In Figure 6, there are nine objects in the project, each of which is represented as a circle. For example, the visitor's profile object stating the visitor's current preferences supports a number of methods for returning the visitor's details. Each object notifies other objects by calling its methods, *e.g.* the *returnPreferredLanguage* method of the visitor's profile object can be invoked by either the control object, the tour creator object or the local Web service object.

Besides the GUIDE project, cues [SBG99] developed with the TEA project is another representative of this model. Both of the two approaches manage

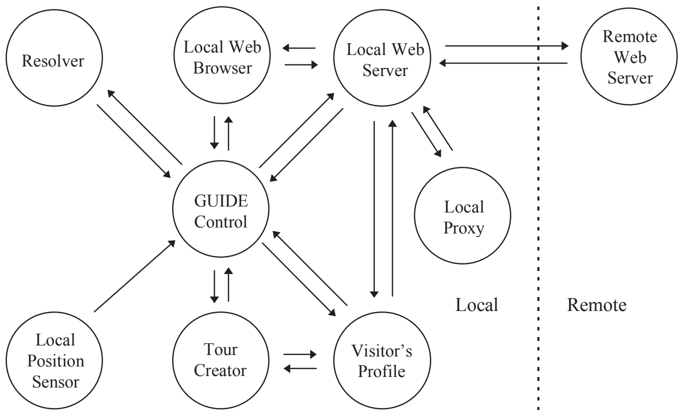


Figure 6: The GUIDE object model [CMD99]

context information into active objects such that details of the data are encapsulated within objects and invisible to other components of context-aware application systems.

- Logical Based Models

  Logical Based Models define the context as facts, expressions and rules using logic theory. Any operation on the context information, such as adding, deleting or updating, is based on logic reasoning. Formalizing Context is the first logic based context modeling approach [McC93, MB97]. Both [McC93] and [MB97] formalize context as abstract entities with properties such that axiom can be applied. [McC93] defines the basic relation $ist(c, p)$, which asserts that the *proposition* p is true in the *context* c. The formulas associate the propositions true to different contexts. The following example $c0$ asserts that it is true in the context of the Sherlock Holmes stories that Holmes is a detective [McC93].

  $$c0: \; ist(context\text{-}of(\text{“Sherlock Holmes stories”}), \text{“Holmes is a detective”})$$

  [McC93] also defines other useful relations among contexts. Suppose $specializes\text{-}time(t, c)$ is a context related to c where the time is specialized to have the value t. Then, we may have the following relation where $at\text{-}time(t, p)$ asserts that the proposition p is true at time t.

  $$\begin{aligned} c0: \quad & ist(specializes\text{-}time(t, c), at(jmc, Stanford)) \\ & \equiv ist(c, at\text{-}time(t, at(jmc, Stanford))). \end{aligned}$$

- Ontology Based Models

  Ontology describes knowledge as concepts and relations, which is especially suitable to model information in certain domains. There are several reasons to model context based on ontology. One reason is that the use of context ontology eases knowledge sharing since context ontology defines a common set of concepts about context. The other reason is knowledge reuse in the sense that developing large-scale context ontology can reuse well-defined Web ontology of different domains rather than start from scratch.

  A representative work of this model is the ontology-based formal context model CONON [WZGP04]. Since formalizing all context information is likely to be an innumerable task, the CONON model captures four types of fundamental context information, namely location, user, activity, and computational entity. In order to provide a flexible interface for domain-specific knowledge, CONON designs a two-level ontology. The upper ontology captures the general feature of context entities. An example of the upper ontology is presented in Figure 7. The conceptual object includes *Person*, *Activity*, Computational Entity (*CompEntity*) and *Location*, as well as a set of sub-class. The lower ontology is a collection of ontologies that describe the details of general concepts and their features in each sub-domain. Figure 8 gives an ontology in the home domain, where several concrete sub-classes are defined to model specific context in a given domain, *e.g.* the abstract class *Device* is classified into sub-classes *TV*, *DVDPlayer*, and *CellPhone*. When modeling context based on an ontology, context can be processed with logical reasoning mechanisms. Context reasoning is implemented in CONON model in two ways: ontology reasoning using Description Logic (DL) and user-defined reasoning using first-order logic. DL reasoning fulfills important logical requirements including concept satisfiability, class subsumption, class consistency, and instance checking. User-defined reasoning aims at defining user-defined context reasoning rules to derive user's situation.

  There are also other research work on ontology-based context model. Ötztürk and Aamodt [zA97] proposed an approach of ontology based modeling. They take advantage of their strong background knowledge on normalization and

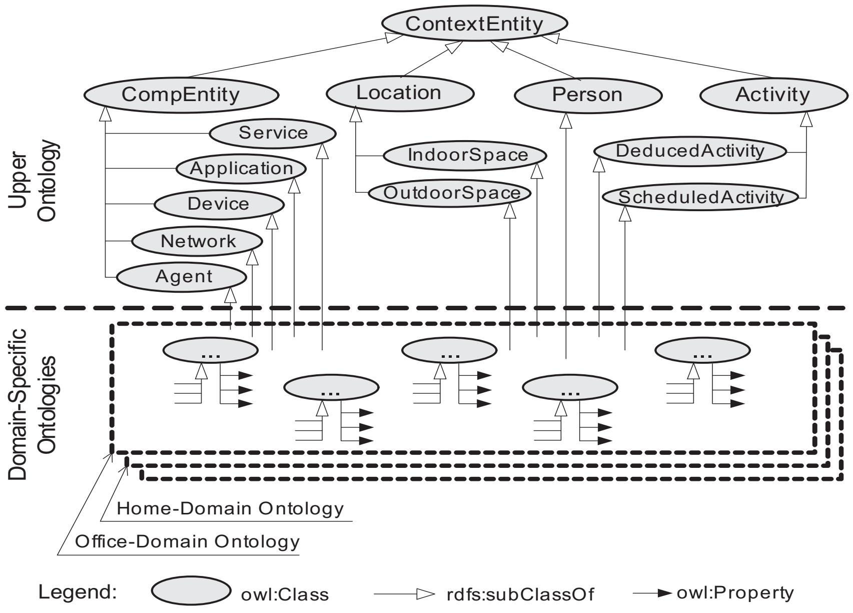


Figure 7: Partial Definition of CONON upper ontology [WZGP04]

formality to study the difference between recall and recognition. *Aspect-Scale-ContextInformation(ASC)* model [Str03] is another approach, which models the context as concepts, sub-concepts and facts and the context knowledge is evaluated using ontology reasoner. *CoBrA* [CFJ03] adopts OWL-DL ontology model for context-awareness.

2.2.2 Mobile Computing

Human-computer interaction is divided into a number of different fields, including that of mobile computing [CW10]. This subdivision of human-computer interaction involves the computer frequently being transported as part of its normal use. Mobile computing involves software and hardware. Mobile computing frequently involves communications as well. There are a number of different devices used for mobile computing. These devices include tablet computers, personal digital assistants, iPads, laptop computers, and smart phones. The smart phones are now the most common form of mobile computing [Gol11].

Another problem with mobile computing is a lack of bandwidth [BKH11]. Accessing the Internet through a mobile device such as a smart phone is usually quite a bit slower than a cable connected desktop computer. However, some of these limitations have been reduced through the use of third-generation mobile telecommunications

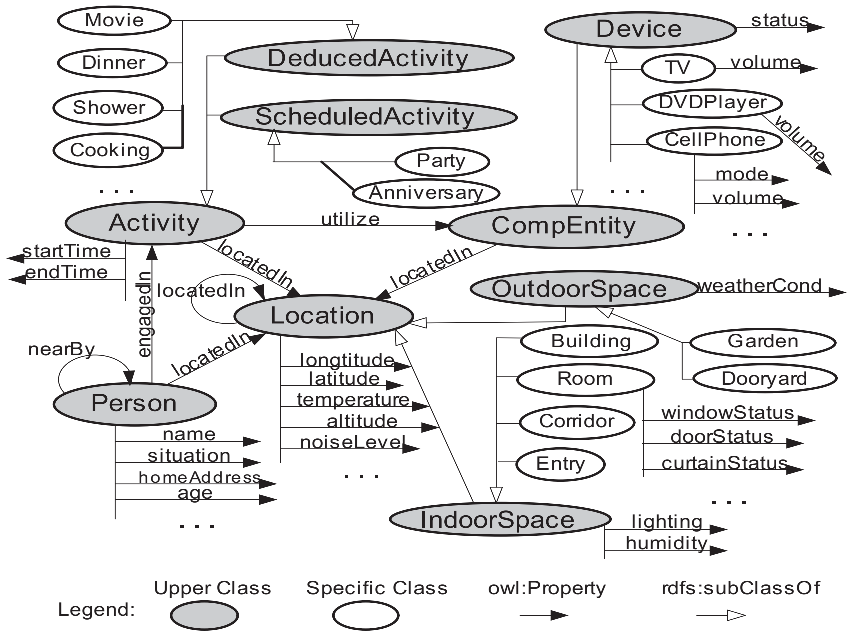


Figure 8: Partial definition of a specific ontology for home domain [WZGP04]

networks (3G) and other solutions such as high-speed uplink packet access (HSUPA). It should be noted here that despite the claims of some telephone carriers that they have 4G networks, this is not actually the case. In the United States, AT&T and Sprint have advanced 3G networks, which approach 4G speeds. Nevertheless, even these networks are quite a bit slower than a computer with a cable connection to the Internet [Pie11].

2.2.3 Smart Phones and Contextual Awareness

Smart Phones

The number of mobile computing devices being used globally is staggering and exceeds 3 billion [Sob10]. The most common type of mobile computing device is now the smart phone. It is estimated that one-third of individuals in the United States have some type of smart phone. Similar figures exist for countries in Western Europe and other developed areas around the world such as Australia and China [TAY10].

The smart phone is actually a small computer with the ability to make telephone calls and connect to the Internet through mobile broadband and Wi-Fi [Pie11]. The smart phone consists of a mobile computing platform. Some of the common mobile operating systems include the Samsung Bada, Microsoft Windows Phone, BlackBerry

OS, Symbian, Apple iOS, and the Google Android. Nearly all smart phones also have Web browsers, touch screens, GPS navigation, video cameras, digital cameras, media players, and a variety of digital assistants [Pie11].

An important aspect of smart phones is that they have application programming interfaces, which are able to take advantage of third-party applications [Pie11]. This means that the smart phone can take advantage of applications, which are fully integrated with its hardware and operating system. The API can be understood as an interface which allows various software components to trade information. There may be specifications for variables, object classes, data structures, and routines [Pie11].

An API can be language dependent or independent [TAY10]. Service-oriented APIs is an advantage to be language independent. This allows the API to communicate with multiple programming languages. It is then useful for many different types of systems and processes. This can be considered as part of the service composability which was previously discussed within this paper.

Global Positioning Systems and Location Awareness

Nearly all smart phones come equipped with a global positioning system (GPS) [Gol11]. This is a navigation system which is satellite-based. The system provides location information for any place upon the earth when there is a line of site which is unobstructed to at least four GPS satellites. This system is accessible to anyone who has a GPS receiver. The system was developed in 1973, but did not become fully operational until 1994 [Zel10].

The GPS in a smart phone means that the device has location awareness [Pie11]. It can actively or passively determine its location. These devices provide the opportunity for the user to do navigating and locating in real time. This allows for the opportunity of designing applications, which make use of this awareness [BKH11].

Since the smart phones have APIs, which allow for third-party applications, there is a wide variety of applications, which make use of location awareness in smart phones [TAY10]. For example, an individual with an iPhone has a choice of several different applications, which provide directions for driving to a certain location. These applications can be downloaded from the Web directly through the smart phone. The individual can then access the application and enter their destination. Since the telephone has location awareness, there is no need for the user to enter their present location, only the destination. Many of these applications will then provide the user with turn by turn directions while driving. Since the telephone has location awareness, it can tell the driver to turn the proper time. These applications will even recalculate directions if the driver passes the point at which they should have turned [ODW11].

Contextual-aware Computing

Since the turn of the 21st-century, contextual-aware computing has been explored as a way to use situational information [BKH11]. This is done through the use of sensor technologies and mobile computing. Contextually aware computing has a goal of providing people with applications, which are improved. These applications will have superior functionality and the easier to use. Early applications of this approach allowed for computing based on the physical parameters of the user. For example, a contextually aware navigation system could compute the best route of travel taking into account traffic congestion. It might use a route which avoids the most congested part of an urban environment [ODW11].

Contextually aware computing arose out of a communication problem between people and machines such as computers or other complex tools [ODW11]. People interacting with new machines usually only have the provided instructions in order to understand how to interact with the system. They are generally unaware of the internal logic being used by the machine. The machine can only interpret the person's needs according to their direct actions. Traditionally, the machine has no knowledge of the context in which the human's actions are occurring [BKH11].

Part of the solution to the human-machine interaction problem involves making the machine since the context in which the person's actions are occurring [ODW11]. This means the machine must be given the appropriate tools to ascertain the context. These tools may be in the form of sensors, cameras, or GPS systems. The machine must also be designed with artificial intelligence so that it can adapt its output based on the results of previous inputs, outputs, and the result [Zel10].

Smart Phones and Contextual Awareness

An idea which is related to location awareness is that of contextual awareness [Pie11]. This refers to the ability of a mobile computing device to determine the context in which it is being used. The concept originated as part of pervasive computing. A device which has context awareness, such as a smart phone, will be able to detect the circumstances in which it is being used. Smart phones which have contextual awareness can interact more easily with users and provide more relevant information. For example, they are more effective at discovering services, which may be required by the user. The contextually aware device has an interface which is adaptable [ODW11].

In order for a smart phone, or any other computing device, to be contextually aware it must have sensors, which provide the appropriate information [BKH11]. In the case of a smart phone, the GPS system, microphone, and camera can work as sensors for contextual awareness. For example, an individual using an iPhone 4s can simply ask the phone, "What is the weather going to be like tomorrow?" The telephone will then tell the user the weather forecast for tomorrow in the area they are located [iSUG12]. There is no need to inform the telephone of the user's location as it is aware of the location through the GPS system. The microphone has picked up the voice request from the user. This information is subsequently processed by software in the telephone and interpreted. Once the information request is interpreted, the telephone will then gather the information from Web-based weather forecasts.

Siri

An example of an application of contextual awareness with a smart phone is the Siri system available on the most-recent Apple iOS operating system [iSUG12].

This is a personal assistant which is intelligent. There is a user interface which takes advantage of natural language in order to answer questions. Siri will also perform actions, make recommendations, and delegate requests [iSUG12]. The system adapts to individual preferences of the user over time. There is a personalization of results. The system takes advantage of location and contextual awareness. For example, if the user asks Siri, "My car is making a terrible noise under the hood. What should I do?" The answer is likely to be something such as, "I have found five automobile repair shops near you." The screen on the telephone will then list the repair shops and their distance from the user. Tapping on one of the repair shops will result in more information being provided. Tapping on the telephone number of the repair shop will cause the phone to call the number [iSUG12].

Text messaging with Siri is an interesting experience [iSUG12]. If the user states, "Siri, I would like to send a message", the response is likely to be something such as, "who would you like to send a message to?" This is an example of contextual awareness because the smart phone has already detected that no contact has been chosen [iSUG12]. If one first looks up a contact on their smart phone and then says the exact same thing, a different response is given. For example, a user might access a contact named John. They might then say, "Siri, I would like to send a message." This time the response is likely to be something such as "what would you like the message to say?" The phone is aware of the context, and will send a message to John unless otherwise specified [iSUG12].

2.3 Service Mashup

Service mashup refers to making data more useful through aggregation, visualization, and combination [Sob10]. This data can then be more easily used by people in both a professional and personal context. Mashups require that data be accessible on a permanent and consistent basis. For this reason, a majority of mashups are posted online or are part of the client application [Gol11].

There is an increasing trend toward Web applications publishing APIs enabling developers to integrate functions and data instead of building these from scratch [Sob10]. The emerging Web is experiencing an evolution in social software due to the active role of mashups. There are a number of mashup composition tools, which are relatively easy to use. Many of these mashups do not require the user to have programming skills. Instead, the graphical user interface (GUI) consists of widgets. These widgets often respond to the user in a more intuitive manner such as allowing instructions in the native language of the user. In this way, the widget allows the user to interact with the system without having to use a computer programming language [Sob10].

There is a wide range of mashup types [Gol11]. These include data mashups, consumer mashups, and business mashups. The data mashup combines information and media which are similar and from several sources and combines them to form a single representation. This combination of resources means there is a novel Web service which did not previously exist [Gol11].

The consumer mashup uses a combination of data types. Data from numerous public sources are located and organized into a simplified GUI.

The business mashups are frequently applications, which combine an organization's resources, data, applications, and external Web services [Gol11]. This information is presented in a manner, which allows the organization to collaborate with other organizations or experts. These types of mashups are often visually rich applications offered over the Web in a secure fashion.

2.4 Service Composition

2.4.1 The Definition of Web Service Composition

Service composition involves ensuring that several existing services can work together in order to provide a calm positive service which better meets user requirements [Gol11]. The increased interest in this type of composition is due to the emergence of a wide variety of Web services, which can work in conjunction in order to provide unique solutions for consumer needs. Many of the techniques used in service composition consist of composite services and the expression of elemental services. The composite services consist of a number of elemental services and possibly other composite services. The composite services require that there be information, which is controlled and flows between different services [ODW11].

The techniques used in service composition have been derived from languages for workflow and business process modeling [ODW11]. There is also an overlap with certain types of software engineering used in systems assembly. With regard to the systems assembly, it is assumed that components will have services, which are well-defined. The static aspects of service composition are dressed by the Architectural Description Languages (ADLs). These include expression of the position of protocol by using connectors [ODW11].

There are several different approaches used for service composition [Gol11]. Proactive composition is done off-line and is used with stable resources, which are always operational. These platforms are generally rich in resources. Reactive composition is used with compound services, which must be created in real time as part of a composition manager. This approach must often make use of real time parameters. An example of this would be the available bandwidth of the network. There is also a difference between mandatory-composite services and optional-composite services. The mandatory-composite services require that the behaviors of all components be corrected. The optional-composite services do not make this requirement, and some components may not be available for correction [ODW11].

Service composition means that there must be commonality between services [ODW11]. The Web Services Description Language (WSDL) has motivated and increased interest with regard to service interoperability. This language has a binding to SOAP, which is standardized. This means HTTPS and be used as a relatively universal communication infrastructure. Transporting data with XML allows for increased flexibility with regard to transformation and encoding of data [Gol11].

Web services composition refers to the interconnection of Web services, which are used to meet the requirements of certain processes [CW10]. The composition refers to an accumulation of composite and elementary Web services. These services can be combined to serve a function which is beyond the capability of any single component [Gol11]. There are composition rules, which determine the order in which the services are begun. These rules also cover the conditions which invoke the use of such services. An important part of the services used in Web service composition is their ability to be reused for new applications. The composition rules mean that the applications adhere to certain protocols, which allow them to be compatible with other applications [Pie11].

2.4.2 Web Service Composition Modeling

The 21st century service-led economies undergo constant change [Sob10]. This means business processes must be flexible and able to both arrange and combine service systems. The process activities are needed in order for 21st century organizations to work efficiently and with high productivity. This environment has increased the need for Web service composition. Considerable work has been done to combine existing services in order to provide new functionality through a system which allows the services to interact [CW10].

In order for the service systems to be properly integrated modeling must be done [CW10]. One technique for Web service composition modeling uses petri nets. These types of models can be simultaneously validated and verified. Another approach is to use XML nets for the modeling. This is actually a variation of the higher-level petri nets. The XML nets are better at exchanging structured data between processes and describing objects [Sob10].

The XML nets are especially valuable for Web service composition modeling as their messages tend to be more structured [CW10]. These messages can also be used as place tokens in order to accomplish message passing. Message passing is crucial for proper integration of services within a system. The filter schemas used in XML can manipulate the XML-based tokens. This is useful when the constraints or criterion must be modeled accurately [Sob10].

The XML nets are actually high level petri nets within which the places are used as containers for the XML documents [Sob10]. These types of documents must have the XML schemas identified by their place. Predicate logic expressions are used to inscribe transitions. With this approach, the free variables are within the inscriptions. These inscriptions have adjacent edges.

The XML net transition is enabled for a specific marking and instance of variables within the transition environment. However, this is only accomplished when certain conditions are met. Each place within the preset transition must contain a minimum of a single XML document. This document must coincide with the filter schema. Furthermore, there must be places within the post-said transition. Transitions which modify the existing XML documents must be checked for validity. The transition inscription needs to be evaluated as well [CW10].

2.4.3 Web Service Composition as an AI Planning problem

The WSDL files described the interfaces for a Web service [Sob10]. Both the output and input parameters of the service can be defined by messages, which are within the WSTL file. If a single Web service is unable to satisfy the business requirements, the need is created for multiple Web services to meet the need. This means a composition of the services which works together as a system is required. The Web service composition is used to determine which group of interconnected Web services can be utilized in order to form the input of another group of Web services. This will allow the composition request to be met by certain output parameters [Sob10].

The situation previously described is known as a Web service composition problem [Zel10]. These types of problems can be solved using a planning graph [YZ08, ZY08]. This graph makes use of iterative deepening. When this occurs, the iteration finds a new piece of the search space. In other words, the planning graph will expand iteratively by single levels during iteration. This process continues until the proposition set includes all the goal propositions. Alternatively, the process can be discontinued at a predetermined fixed-point level. If the planning problem does not have a solution, the planning graph will contain no output. If a solution is found, the planning graph will describe a sequence of actions, which will result in the problem's solution [Zel10].

Planning graphs can be described by algorithms [Sob10]. These algorithms allow the computer to use a planning graph in an automated fashion. These types of algorithms are crucial factors for successful Web service applications. A wide variety of methods has been used in order to attain this goal. However, the majority of them are either artificial intelligence (AI) planning techniques or workflow-based. Additional methods include finite-state machine techniques, model checking, Petri nets, p-calculus, and algebraic processes [Zel10].

When the workflow-based technique is used, the Web services are conceptualized as similar to the work flows [Zel10]. With this approach, there is control of the data flow by the Web services. There are a number of industry standards, which use this approach such as the Business Process Service mashup refers to making data more useful through aggregation, visualization, and combination. This data can then be more easily used by people in both a professional and personal context. Mashups require that data be accessible on a permanent and consistent basis. For this reason, a majority of mashups are posted online or are part of the client application [Sob10].

2.5 A Motivating Example: Personal Entertainment Planner

2.5.1 Android Phone

Android is a Linux-based operating system for mobile devices such as smart phones and tablet computers. It is developed by the Open Handset Alliance, led by Google, and other companies. Google releases the source code of Android as open source, under the Apache License. Android is becoming a major competitor to iPhone OS in terms of openness. Android has a large community of developers writing applications ("apps") that extend the functionality of the devices. Applications for android devices are written in Java language [Wik12a].

An app on Android OS can access system information. Below is a list of the types of information an app can access [Per10]:

- SD storage - modify/delete SD card contents, such as pictures, videos, mp3s, and even data written to the SD card by other applications.
- Contact list - read/write contact information
- Calendar data - read/write calendar data
- Location - fine (GPS) location, or coarse (network-based) location
- Mails - access preset mail accounts (*e.g.* Gmail accounts)
- Messages - read/send instance messages
- Network communication - full Internet access, view network status, view Wi-Fi state.

Normally, it is forbidden to access information between different applications on Android OS [And12]. However, two ways make destination applications accessible. One is called *SharedPreference* [And12]. In this way, destination applications expose their interfaces such that we could invoke them with a component called *Content-Provider*. The other one is called *SharedUserId* [And12]. In this way, we can access destination applications as our own data by setting with the same *SharedUserId*. Hence, we are able to access the applications on Android OS, if the applications are developed by ourselves or expose their application information in *SharedPreference* way or in *SharedUserId* way. For example, if a weather forecast application exposes its interface, we can call this application to get weather information. Otherwise, it is impossible to access the destination applications. Please note that majority applications, such as social network application, *e.g.* Facebook and twitter, do not provide any way to access their applications' data , which is related with users' private information, such as contact list, on Android OS.

2.5.2 Personal Entertainment Planner

The goal of the Personal Entertainment Planner is to assist the owner in entertainment scheduling / planning in accordance to other determining factors. For instance, when you travel to a new city for a business trip, you would most likely want to entertain yourself most often especially at night after a long day of hard work *i.e.* the business schedule during the day. Now, suppose you carry a smart mobile device (smart phone or tablet) which can be or is connected to the Internet. The device would be more efficient and reliable to you if it could be having an application that can guide and/or control you on how to spend, *e.g.* from 7:00PM to 11:00PM this evening, "I will entertain myself somewhere not far from the current location". The smart mobile device's entertainment planner application will need to use the information stored in the device such as contacts, current locations. The owner of the device is casual about what he or she can do for the evening. The owner is only constrained by time, location, and money. The entertainment planner is expected to give the owner the following options:

Option 1: Dinner at Restrauant L'Autre Saison from 7:00-8:00; Watching movie "The Help"at cinema "Cinema Banque Scotia Montreal"from 8:45-11:00;

Option 2: Dinner at Seven Night Club and watch the Hockey game "Canadien vs. Boston"from 7:30 to 11:00;

Option 3: Dinner at my friend Brian's home from 6:30-7:00; Concert at "Place des Arts", from 8:30-11:00;

The entertainment planner is a composed service for mobile devices. It invokes related data services to get proper information *e.g.* the restaurants nearby, the movie schedules, and driving directions etc. It may access personal information stored in the mobile device for use to enhance its functionality and display them on the application's GUI through the device's screen as List 2.5.1.

The entertainment planner also needs to be smart and flexible. The user can either provide very weak requirements such as to "spend a few hours in downtown Montreal" or provide more specific requirements such as, the category of activities, the starting time and duration, budget, etc. The entertainment planner should be able to search for all the possibilities at the local place and make a plan for the user.

Chapter 3

Constraint Model for Context Representation

3.1 Context

Context is defined 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 mobile application, entities are the knowledge that can either be automatically detected by the mobile system, *e.g.* the location, the identity and the current time (see Section 2.2.3), or provided by the user *e.g.* the budget and the entertainment activities.

There are several context modeling approaches for context representation including key-value models, markup schema models, object oriented models, graph models, ontology models, and logic models (see Section 2.2.1). In this section, we propose a constraint model for the description of various contexts in our mobile application. Our context model can be regarded as key-value model plus inequalities. Though based on defined ontology, our context is different from ontology model, where ontology models reason over low-level context to drive high-level context using logical reasoning, but there are no reasoning tools embedded in a mobile phone. In our work, we expect our context models have the expressive power to handle both facts and constraints assigned with real values, boolean propositions, and inequalities. Also, our context model is expected to be handled by a compositional algorithm. Therefore, we propose a constraint model for our context representation.

Firstly, we define a domain related ontology shown in Table 2.

Table 2: Ontology for Entertainment Planner (also serve as domain terminology)

type	subtype	sub-sub-type	attributes
Location			
	Current location		
	Destination		
	Start location		
	Distance		
Money			
	Cost		
Time			
	Time point		
		Start time point	
		End time point	
	Duration		
Entertainment			
	Movie		
	Dining		
Service			
	Movie Theater		
	Restaurant		
Self Service			
	Driving		
Personal Identity			
	Me		
Activity			ActivityName, Cost, Duration, Distance, StartLocation, Destination, MovieName, StartTimePoint
Name			
	ActivityName		
	MovieName		

Table 2 defines the ontology for the entertainment planner, which serves as domain terminology for the motivating example presented in Section 2.5.

The context information is represented by types in the Table 2, which includes location, money, time, entertainment, service, personal identity, activity and name. A type can be a subset of another type. For example, *Current location* is a subtype of *Location*. The attributes of an entity are defined as a set of types. The type *Activity* has the attributes of *ActivityName*, *Cost*, *Duration*, *Distance*, *Start Location*, *Destination*, *MovieName*, and *StartTimePoint*.

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

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

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

dataType := real|nature number|integer|boolean|string.

ontologyType is a domain related ontology. We can find the corresponding values of *ontologyType* in Table 2.

Example 1 *m is used to represent the money the device owner currently has. Then,* $variableName = m$*,* $dataType = real$*, and* $ontologyType = Money$.

We define all variables in Table 3 according to the Definition 1.

Table 3: Variables

variableName	dataType	ontologyType	variableName	dataType	ontologyType
l	string	Location	ds	real	Distance
m	real	Money	c	real	Cost
t	Date	Time	d	real	Duration
mv	boolean	Movie	dn	boolean	Dining
mt	boolean	Movie Theater	r	boolean	Restaurant
dr	boolean	Driving	p	boolean	Personal Identity
i	boolean	Me	a	string	Activity
an	string	ActivityName	mn	string	MovieName

Definition 2 *A* ***constraint*** *is represented as an equality or an inequality of variables.*

An equality gives the value of a variable. For example, current location $l =$ "1455 De Maisonneuve West". An inequality gives a range of the value of a variable. For example, we can define a location l' should be within 5 kilometers of current location l as $|l - l'| < 5$.

For the convenience of describing the constraints, we can use the approximation operators in Table 4 in our application. User can select approximation operators to give the constraints.

Table 4: Approximation Operators Used in Inputing Data (x is a variable)

Operator	values	equality&inequality
near	V, L	$\|x - L\| < V$
less than	V	$x < V$
greater than	V	$x > V$
all	$V1, V2, V3, ...$	$x1 = V1 \wedge x2 = V2 \wedge x3 = V3$
any of	$V1, V2, V3, ...$	$x = V1 \vee x = V2 \vee x = V3$

After defining variables and constraints, we can define a context as below:

Definition 3 *A* ***context*** *X is a set of constraints about the variables.*

The constraints in a context can be over both proposition symbols and real value variables. Proposition symbols take two values $true$ or $false$. The constraints over proposition symbols can be used to describe the status of a system. For example, "entertainment=$true$"means the user have done the "entertainment"already. Otherwise, the user does not have any "entertainment"yet. Also, the constraints can be over real value variables, *e.g.* location, cost and duration.

Example 2 *Suppose a user carries a mobile device. Then a context example is as in Table 5*

Table 5: A context example

variable name	data type	ontology type	constraints
l	string	current location	$l = L$
t	string	current time	$t = T$
m	real	possessed money	$m = C$
mv	boolean	movie	$mv = false$

From the current context in Table 5, we know that the user is currently at location L with C amount money in his pocket or in his online account. He has not watched any movie yet. The current time is T.

One important characteristic of context-aware system is that actions that can be preformed on the current context are algorithmically defined. We give the definition of an **activity** as follows.

Definition 4 *An* ***activity*** *a is a tuple $\langle Pre_a, Attr_a, P_a^+, P_a^- \rangle$, where Pre_a, P_a^+, P_a^- are finite set of constraints. Pre_a is a set of preconditions, P_a^+ and P_a^- are positive effects and negative effects respectively, and $Attr_a$ is the attributes of a.*

The attributes $Attr_a$ of an activity a are described in Table 2, *e.g.* Cost C_a, Duration D_a, Start Location $L1_a$, Destination $L2_a$, and StartTimePoint T_a (used for movie activity).

For an activity a, Pre_a, P_a^+ and P_a^- specify the constraints about the variables. Pre_a is the precondition of a containing a set of constraints over propositional symbols. P_a^+ and P_a^- represents the positive and negative effects after performing activity a respectively. The constraints in both P_a^+ and P_a^- are over proposition symbols. We can see an example of a dining activity a in the Equation 1.

$$\begin{aligned} a = \; & \langle \{dining = false\}, \{L1_a = \text{“2501 Notre Dame, Montreal, QC, CA, H3J1N6”}, \\ & L2_a = \text{“2501 Notre Dame, Montreal, QC, CA, H3J1N6”}, D_a = 120, C_a = 25\} \\ & \{dining = true\}, \{dining = false\} \rangle \end{aligned} \tag{1}$$

Equation 1 shows a dining activity. Pre_a is a set of constraints. $dining = false$ means the activity a can be applied only if the user does not have dinner yet. Both the start location $L1_a$ and the end location $L2_a$ are “2501 Notre Dame, Montreal, QC, CA, H3J1N6”. Please note that if a is a driving activity, then we have $L1_a \neq L2_a$ because people must drive from one place ($L1_a$) to a different place ($L2_a$). For any other activities, for example, movie activity and dining acitivity, we have $L1_a = L2_a$ because people must stay in the same place to watch a movie or have a dinner. $C_a = 25$ means the cost of activity a is 25 dollars. $D_a = 120$ means the duration of the activity a is 120 minutes (2 hours). $\{dining = true\}$ and $\{dining = false\}$ are the positive effect P_a^+ and the negative effect P_a^- respectively, which shows how the problem space is affected after activity a is performed.

3.2 Use Cases of the Context Model

Example 3 *Suppose the same user in Example 2, wants to spend the next D hours entertaining him or herself around the current location. With a proper entertainment* *planner's UI, the inputs from the user can be translated to the initial context and goal context as in Table 6. The current context at any given time is also shown in Table 6.*

According to the context in Table 6, the user is currently at location L with C amount of money in the pocket or in his or her online account. The current time is T. The user wants to spend D hours having a dinner and watching movies nearby, and does not mind driving from one place to another. The total expense is not supposed to exceed C dollars. At any given time in the system, the current context is always satisfying the distance requirement,*i.e.* the radius is within R kilometers, *i.e.* $R \leq 5$, the available money *i.e.* $m \leq C$ (the expense never exceeds the allocated budget) and the time should not exceed D hours.

Table 6: A context example - initial context, current context and goal context

variable name	data type	ontology type	approximation operators	value	constraints
init-context					
l_i	string	start location		"L"	$l_i =$"L"
m_i	real	possessed money		C	$m_i = C$
t_i	real	start time		T	$t_i = T$
current context					
r	real	location range		R	$r = R$
l	string	location	near	any location point	$\|l - l_i\| < R$
d	real	duration		D	$d = D$
t	real	time	within	any time point	$t - t_i < D$
m	real	possessed money	greater than	any amount of money	$m > 0$
goal-context					
l_g	string	destination		"L"	$l_g =$"L"
m_g	real	possessed money	greater than	0	$m_g \geq 0$
t_g	real	end time	near	T,D	$t_g - T \leq D$
mv	boolean	movie		true	$mv = true$
dn	boolean	dining		true	$dn = true$
dr	boolean	driving	any of	true,false	$dr = true \vee dr = false$

3.3 Context Operations and Reasoning

There are also relations among different contexts. In this section, we define the operations between different contexts and the reasoning that we deduce from one context to another context.

Definition 5 *Suppose X is the current context. There exists an activity a* ***applicable*** *to X, denoted as $X \succ a$, if the following conditions are satisfied:*

- *Start location satisfaction: $l = L1_a$, where l is the location of X and $L1_a$ is the start location of a;*
- *Start time satisfaction: $t \leq T_a$, where t is the time of X and T_a is the start time of a;*
- *Cost satisfaction: $m \geq C_a$, where m is the money of X and C_a is the cost of a;*
- *Constraints satisfaction: $Pre_a \subseteq P_X$, where Pre_a is the precondition of activity a and P_X is a set of the constraints over proposition symbols in X.*

Example 4 *Suppose there is an activity a, called $WatchMovie$ presented as below:*

$$\begin{aligned} a = \ & \langle \{movie = false\}, \{L1_a = \text{“}L\text{”}, \\ & L2_a = \text{“}L\text{”}, D_a = D, C_a = C, T_a = T\} \\ & \{movie = true\}, \{movie = false\} \rangle \end{aligned} \tag{2}$$

Currently, the user is at location l with c amount of money in his pocket, where $l = L$ and $c > C$. The current time t is earlier than the start time of the movie, i.e. T, and the user has not watched movie yet. We can see the activity a is applicable to current context X, i.e. $X \succ a$.

Definition 6 *Suppose X is the current context. If there exists an activity a and $X \succ a$, a new context X' is transformed from the current context X when applying the activity a on the current context. The procedure is denoted as $X \overset{a}{\Longrightarrow} X'$. The context X' satisfies the following constraints.*

- *$l' = L2_a$, where l' is the location of X', and $L2_a$ is the destination of a;*
- *$t' = T_a + D_a$, where t' is the time of X', T_a is the start time of a, and D_a is the duration of a;*
- *$m' = m - C_a$, where m' and m are the money of X' and X respectively, and C_a is the cost of a;*

- $P_{X'} = P_X + P_a^+ - P_a^-$, *where* $P_{X'}$ *and* P_X *are the set of constraints over proposition symbols in* X' *and* X *respectively.*

Example 5 *According to the Example 4, we can apply the reasoning, i.e.* $X \overset{a}{\Longrightarrow} X^a_{post}$, *to get a new context* X^a_{post}, *which is post-context of activity* a. *Table 7 lists the pre-context and the post-context of* a.

Table 7: A $WatchMovie$ activity - pre-context and post-context

variable name	*data type*	*ontology type*	*constraints*
pre-context			
l	*string*	*current location*	$l =$ "L"
t	*date*	*current time*	$t = T'$
d	*real*	*duration*	$d = D$
m	*real*	*possessed money*	$m = C'$
w	*boolean*	*movie*	$w = false$
post-context			
l	*string*	*current location*	$l =$ "L"
t	*date*	*current time*	$t = T + D$
d	*real*	*duration*	$d = D$
m	*real*	*possessed money*	$m = C' - C$
w	*boolean*	*movie*	$w = true$

3.4 Comparison with Existing Contexts

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

- Key-value models use key-value pairs to model the context. The key-value pairs define the properties and the values of the environment. Context-aware applications using key-value models do not implement complex reasoning; they implement only simple reasoning, *e.g.* IF-THEN rules. Also, Key-value pairs are easy to manage, but lack capabilities of handling complex context information. The context information in our work is more complex than key-value models, since inequality is also included in our context model. Hence, key-value model is not used.

- Markup schema models define the context into a hierarchical data structure consisting of markup tags with attributes. The attribute of the markup tags is recursively defined by other markup tags. There are no constraints defined in markup schema models to describe the value of markup tags. However, in our context modeling, it is necessary to define the constraints for the description of variables values. Thus, our context modeling does not make use of Markup schema models.

- Object oriented models aim at taking the benefit of the encapsulation and reusability of the object oriented approach. The process of managing the context is encapsulated on an object level. The specified interfaces are defined to have access to the context information. In our context modeling, it is unnecessary to hide the details of context processing. Thus, our context modeling does not make use of object oriented models.

- Graph models are extensions of object-oriented models. Graph models use graphs with graphical notations to represent context information and the dependency relations between classes / entities. However, graph models do not assign the constraints with real values in contrast to our context modeling. As a result, our context modeling does not make use of Graph models.

- Ontology models construct a context ontology using the Web Ontology Language (OWL). Both the graph models and ontology models can translate context information into DL. In ontology models, context reasoning is implemented by DL reasoning to fulfill logical requirements. This model cannot handle constraints assigned with real value but only facts. We need a context model to handle both facts and constraints assigned with real values. As a result, our context modeling does not make use of Ontology models.

- The objective of Logic based models is to propose an adequate theory of reasoning with contexts. However, there is no reasoning tool for mobile phones. Therefore, we do not consider logical based models in our context modeling.

Apart from the existing context modeling approaches, we use constraints in regard to the ontology to describe our context. The characteristics 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 models, the operations to be executed are based on several predefined rules. Also, there are no reasoning tools embedded on mobile phones. In addition we exclude logic rules in our constraint models.

- Expressive power

  We want to handle both facts and constraints assigned with real values, boolean propositions, inequalities etc. Our constraint model defines constraints consisting value assignments and inequalities. Moreover, it is easy to implement this context in programming.

- Extensibility

  We want a model that can be used in any domain. In order to achieve this purpose, we need to get the related ontology for the given domain. And then define a set of variables. Based on them, the context model can be built to fulfill the business goal.

- Algorithmic Processing

  When we implement service composition using a search-based algorithm, we define the constraints to be satisfied as the pre-condition to perform an operation. It is easy for a search algorithm to process constraints consisting of value assignments and inequalities.

Chapter 4

Service Discovery Mashup

Through the smart phone UI, user can input business goals, the business goals can be presented as constraints. Based on the business goals, we can discover related services. The discovery of services paves the way to service composition for achieving the business goals. What we want to do:

- We discover the real world services through querying online resources, including common search engine, RESTful data services rather than UDDI in our work.
- In order to demonstrate the wide coverage of our service discovery technique, we use different types of online resources. MS Bing service is picked as a representative of SOAP search engine to search for business services, *e.g.* restaurant. Google Maps Web service as a representative of RESTful search engine is to find driving direction from the original place to the destination. Driving can be considered as a self-service. Google show time as an HTML engine returns movie services. These services are hand picked to cover all the three types, rather than the merits of their functions.
- We translate the current context and the business goal context into query strings and query criteria.
- We use a mashup to integrate the above different type of services. The mashup invokes the services and collects the query outputs for the service composition procedure. After the mashup procedure, services are turned into activities which can be used in service composition algorithm.

4.1 Service Discovery in Practice

UDDI is a XML based registry to be used by businesses throughout the world in order to list their organization on the Internet and also serves to locate and register Web service applications [Sob10]. Though UDDI is designed for service discovery, it is practically dead with no public UDDI server available. So currently, we use general purpose search engines such as Google [Goo12a], Bing [Mic12a] or Yahoo [Yah12] to discover services.

There are also Web service search engines, for example as following, which collect information about SOAP and RESTful services, *e.g.* the URL, implementation techniques, query format, query examples.

- Woogle [DHM+04] employs clustering technique to search the desired Web services that satisfy user requirements described as keywords. Woogle obtains Web services through Web service description registered on UDDI.

- Seekda [See12] uses the general purpose search engine Alexa to discover Web services. The returned Web services are ranked according to not only the similarities to the users' requirement but also qualities of services, *e.g.* service response time and service reliability.

- WSExpress [ZZL10] acquires Web service description through crawling the Web. The search results are ranked according to both functional and non-functional characteristics.

- SWSE [MSXZ10] and OSSE [Dan10] try to search Web services with higher recall and precision by taking semantics retrieved from ontologies into consideration.

In this thesis, we want to use services which are real world businesses *e.g.* restaurants, movies, more than electronic services in SOA domain, to achieve our business goals. Therefore, we do not use any existing Web service indexing services to discover services. Instead, we use general purpose search engines to discover services and businesses. Then, the discovered services, which are also search engines, are used to discover services participating in service composition. In order to make different types of search engine services work together, we propose a mashup phase to uniform the format of inputs and the output of search engine services.

4.2 The Available Search Engine Services

Our service discovery engine takes the advantage of the use of online data services discovery, business service, and self-services *e.g.* driving.

In order to demonstrate the wide coverage of our technique, we use different types of online data services. More specifically, the search engines are SOAP services, RESTful services, and HTML services. These services are hand picked to cover all the three types, rather than the merits of their functions.

As their request and response formats are similar over the HTTP protocol, it is possible to parse them in a uniformed way (*e.g.* using XML and XPath). Therefore, we use mashup to enable different types of services to work together. The service mashup engine is build by hand.

The inputs of the service mashup engine is a goal context and the outputs of the service mashup engine are a list of services that can be used in the service composition engine.

4.3 SOAP Services and MS Bing Services

The SOAP services provide an endpoint that exposes a set of operations to be invoked by the clients. The endpoint and the operations are described in a standard WSDL [W3C07b] document. A client can invoke an operation by sending the service an XML-based SOAP message. And the client receives an output SOAP message as the response when defined. Purely based on the syntax definitions in WSDL, automatic approaches are developed to compose/parse SOAP messages and invoke services automatically. The semantics of the operations are not explicit in WSDL. And sometimes, additional documents, *e.g.* OWL-S [W3C04a] and SAWSDL [W3C07a], are used to define the scope, effects, pre-conditions and assumptions made by service designers.

The Microsoft Bing SOAP Services is a set of Web services that enables users to search, discover, explore, plan, and share information about specific locations [Mic12b]. There are four Bing Maps SOAP Services: Geocode Service, Imagery Service, Route Service, and Search Service. Search is the operation in the WSDL file for the Search Service. Search returns a parsed query or search results for a given search input string. The request and the response of this operation are SearchRequest and SearchResponse respectively.

Table 8: The MS Bing Maps Search Service

Operation: Search()
Input Parameter: SearchRequest
SearchRequest.Credentials
SearchRequest.Query
SearchRequest.SearchOptions.Count
SearchRequest.SearchOptions.StartingIndex
SearchRequest.SearchOptions.ListingType
SearchRequest.SearchOptions.Radius
SearchRequest.SearchOptions.SortOrder
SearchRequest.SearchOptions.Filters
Output Parameter: SearchResponse
SearchResponse.ResultSets[i].Results[j].Name
SearchResponse.ResultSets[i].Results[j].Address
SearchResponse.ResultSets[i].Results[j].Distance
SearchResponse.ResultSets[i].Results[j].PhoneNumber
SearchResponse.ResultSets[i].Results[j].Categories
SearchResponse.ResultSets[i].Results[j].UserRating

Table 8 lists some of their properties used in our application. `SearchRequest.Credentials` identifies the requestor. `SearchRequest.Query` is a string containing the query to parse and match to a search result. `SearchRequest.SearchOptions` is an object to refine the search request. Useful settings inside `SearchRequest.SearchOptions` are `Count` an int specifying the number of search results to return, `StartIndexing` an int specifying the zero-based array index of the first result to return, `ListingType` an enumerable value identifying the listing type to search, *e.g.* "`Business`" for business listing, `Radius` a double specifying the radius of the circle in which to search, and `SortOrder` an enumerable value specifying how to sort the search results, *e.g.* . by "`Relevance`", "`Rating`", and "`Distance`". In addition, `Filters` specifies how to filter the search results that are returned, *e.g.* `Filters.Category` = 11168 is to search restaurants.

`SearchResponse` contains the results returned by the Search Service. Inside `SearchResponse`, `SearchResultSets` is an object, each element of which contains search results from a specific listing type. In a `SearchResultSets` element, `Results` contains the search results. A `Results` element has the properties like `Name`, `Address`, `Distance`, `PhoneNumber`, `Categories`, and `UserRating`.

Example: Table 9 shows partially the search results for restaurants and coffees around the downtown campus of Concordia University. Input SeachRequest.Query is "Concordia University, Restaurant".

Table 9: Restaurant and Coffee near Concordia University

Name	Address	Distance	Category
Angela Pizzeria & Restaurant	1662 boul de Maisonneuve O, Montreal, QC, H3H 1J7	0.022	Pizza
Tapioca Thé	1672 boul de Maisonneuve O, Montreal, QC, H3H 1J7	0.037	Restaurants
Restaurant Antep Inc	1626 boul de Maisonneuve O, Montreal, QC, H3H 1J5	0.037	Restaurants
Magic Idea	1675 boul de Maisonneuve O, Montreal, QC, H3H 1J6	0.04	Restaurants
Presse Café	1805 boul de Maisonneuve O, Montreal, QC, H3H 1J9	0.078	Coffee & Tea

Convert goal context into query inputs. Input SeachRequest.Query is location + goal propositions, *e.g.* "Concordia University, Restaurant".

4.4 RESTful Services and Google Maps Web Service

Representational State Transfer (REST) services are gaining attentions as a lightweight approach for the provision of services on the Web. The RESTful services are considered as a part of SOA. Unlike WSDL-based services, the central elements in REST are the resources, which are abstract entities identified by URIs that can be manipulated through a set of standard HTTP operations, namely GET, POST, PUT, and DELETE. Resource's state is transferred to/from the clients as the consequence of executing the standard operations. The state is portrayed to the clients by means of representations, which are documents serialized according specific media types (*e.g.* XML, HTML, JSON, text, etc.), and contain hyperlinks to related resources.

Table 10 shows the typical request and response to a RESTful service. The request is to get the customer information for a customer whose id is 1. The URI of the resource is `example.com/resources/customers/1/`. The request indicates the response should be in XML format. The request is sent by HTTP GET. In

Table 10: 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: Tue, 08 May 2007 16:41:58 GMT Server: Apache/1.3.6 Content-Type: application/xml; charset=UTF-8 <?xml version="1.0"?> <customer xmlns=". . . "> <customerID> . . .</customerID> . . . </customer>

the response following, we can see the XML formatted customer information. Many RESTful services, *e.g.* delicious.com [Del12], can also accept query strings. A query string is in the format of "`?variable=value1&variable2=value2`". At the server side, the query string is parsed to map to resources query.

As RESTful services do not have a uniform standard to describe their input and output, the programmer needs to program the code to generate the input query and interpret the output message.To do this, the RESTful services need to have development documents in natural language for the programmers to use.

Google Maps Web Services [Goo12b] use HTTP requests to specific URLs, passing URL parameters as arguments to the services. The output of the services are in either JSON or XML for parsing by your application. A service request is of the following format:

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

In the above HTTP request, service indicates the particular service requested, *e.g.* directions or geocode, and output indicates the response format, which usually is JSON or XML. Some examples of parameters are listed in Table 11.

Example: A request `http://maps.googleapis.com/maps/api/directions/xml?origin=H3H2P3&destination=H3N1G3` is for searching driving directions between two places (Zip code H3H2P3 and H3N1G3), and not using sensor (*e.g.* a GPS as location sensor). The Directions API can return multi-part directions using a series

Table 11: Google Maps Parameters

parameter name	meaning
origin	textual latitude/longitude value
destination	textual latitude/longitude value
mode	transport mode, *e.g.* driving
waypoints	alter a route
alternatives	provide more than one route
avoid	avoid features, *e.g.* tolls
sensor	whether use a location sensor
...	

of waypoints. The structure of the XML is listed as below:

```
<DirectionsResponse>
  <status>OK</status>
  <route>
    <leg>
    <step> ... </step>
    <step> ... </step>
    <duration> ... </duration>
    <distance> ... </distance>
    <html_instructions> ...
    </html_instructions>
  </route>
  ...
</DirectionsResponse>
```

Convert goal context into query inputs. Query string is origin=l&destinatio-n=l', where l and l' are two locations.

4.5 HTML Services and Google Show Time

HTML Services. We generally call all the Web sites that can return HTML pages as HTML services. As they are the most widely used services, we would like to study how to use the information returned from them to serve business goals in service composition. The HTML services take HTTP query (URL+query string) as input and generate HTML pages as output (see Table 12). Comparing Table 10 and Table 12, one can see that from the HTTP operation point of view, a RESTful service is a special sub-case of an HTML service. Their queries are a HTTP query, except a HTML service returns a HTML page, while a RESTful service can return multiple data formats, including HTML. Assuming the content structure of the HTML pages are stable, we can program a parser to process the output HTML pages and get useful information from them.

Table 12: The Request and Response of a HTML Service

Request:
GET /somedir/page.html HTTP/1.1
Host: example.com
Accept: text/html,application/xhtml+xml

Response:
HTTP/1.1 200 OK
Date: Tue, 08 May 2007 16:41:58 GMT
Server: Apache/1.3.6
Content-Type: text/html; charset=UTF-8
<html>
<head> . . .</head>
<body> . . .</body>
</html>

Google Show Time is part of Google search engine. You can send an HTTP query like `http://google.com/movies?near=h3h2p3` to get movie schedule near the location you put in the query string. The returned response is in HTML format as listed below:

```
<div id="movie_results">
  <div class="theater">
  <h2 class="name">
    IMAX TELUS Montreal Science Centre
  </h2>
  <div class="info">
    King Edward Pier, Montreal, QC,
    Canada - (514) 496-4629
  </div>
  <div class="showtimes">
    <div class="movie">
      <div class="name">
        Born To Be Wild IMAX 3D
      </div>
      <span class="info">
        40min-Rated G-Documentary
      </span>
      <div class="times">
        Dubbed 12:10 2:20 3:25 5:35
        7:45 10:00 6:40
      </div>
      ...
    </div></div></div>
</div>
```

4.6 Service Discovery Mashup

We use a class diagram as shown in Figure 9 to illustrate mashup process. The mashup consists of three parsers, *i.e.* *SOAPParser*, *HTMLParser*, and *RESTfulParser*. Every parser inheriting the super class called *ServiceParser* searches its corresponding service, *i.e.* SOAP service, HTML service, or RESTful service, and parses the result of the service into a list of corresponding activities, *i.e.* *SOAPActivity*, *HTMLActivity*, or *RESTfulActivity* in a uniformed format, where those activities inherit the super class *Activity*. As for the *SOAPParser*, MS Bing search returns a response object. The object is parsed and its properties are converted into activities. Also, as for *HTMLParser*, the returned HTML page from Google movies showtime HTML service is parsed and converted into activities. Similarly, the JSON returned by Google maps in *RESTfulParser* is parsed and converted into activities. The generated activites are part of the inputs of the planner.

Currently fully automatic mashup is not possible in the current technique level. The control flow of our mashup shown in Figure 10 is the result of manual work. The control flow begins with searching for restaurants and movies by invoking MS

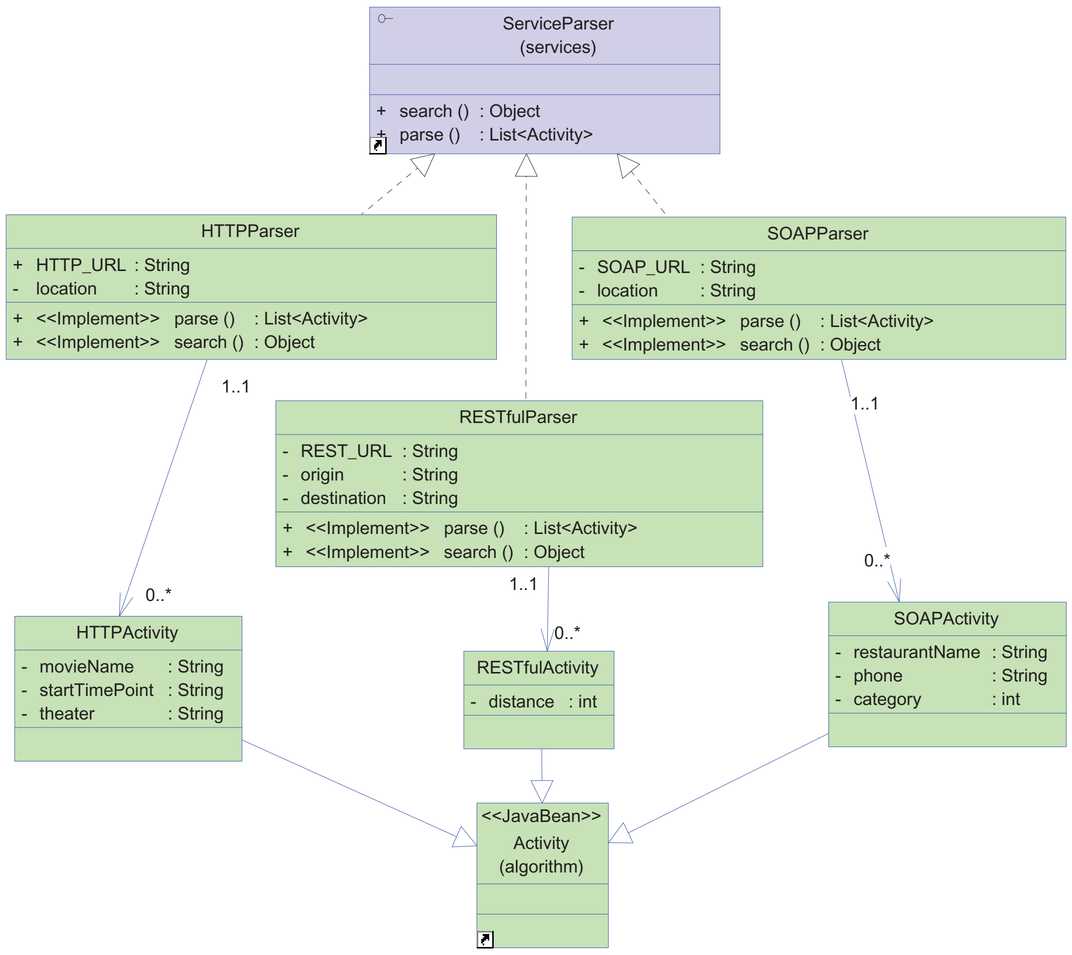


Figure 9: The diagram of mashup

Bing business search (SOAP service) and Google movies showtime (HTML service). These two actions can be executed in parallel. The mashup engine can compose the query request for a service from the query response of another service. For example, we use Google Maps service to get the driving directions. The origin and destination addresses are from the query results of MS Bing Search and Google movies showtime.

Figure 11 shows data flows between the various input and output parameters of the mashup process. The user inputs include the following parameters: “start time”, “end time”, “start place”, “end place”, “budge”, “range”, and “activities”. During the mashup process, the only parameter we use is “start place”. “start place” is recognized as “Location” as shown in Figure 11 to search restaurants and movies around.

Use MS Bing business search as an example. The values for the data fields “name”,

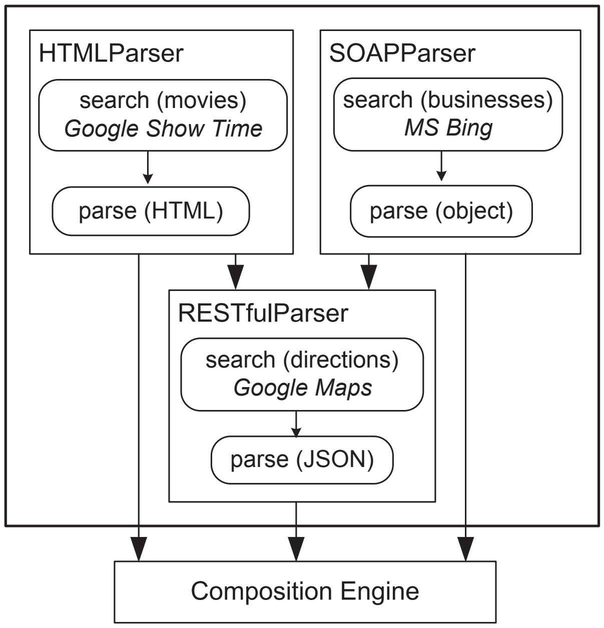


Figure 10: The control flow of mashup

"categories", "address", and "phone number" are extracted from the response object and are used to generate activities to be used by the composition engine (how to generate activities is described in Chapter 6). "Location" is also used as the input to Google Maps search.

Please note, we are not limited to use MS Bing business search and Google movies showtime as our data services. For example, we can add a data service "search events" from Tourisme Montreal events [Mon12] to our mashup process as shown in Figure 11 framed by dotted lines. Tourisme Montreal is also an HTML service and provides plenty of events in Montreal . Once we tell a user is located in Montreal, we can send an HTTP query like `http://www.tourisme-montreal.org/What-To-Do/Ev-ents` to get events in Montreal today. Similarly, we can have the events in Toronto from `https://wx.toronto.ca/festevents.nsf/` if the user is located in Toronto. Moreover, we can add a service "VisitFriends"by getting the information about the addresses of the user's friends from the contact list of the user's smart phone, then use their addresses as the input to Google Maps search.

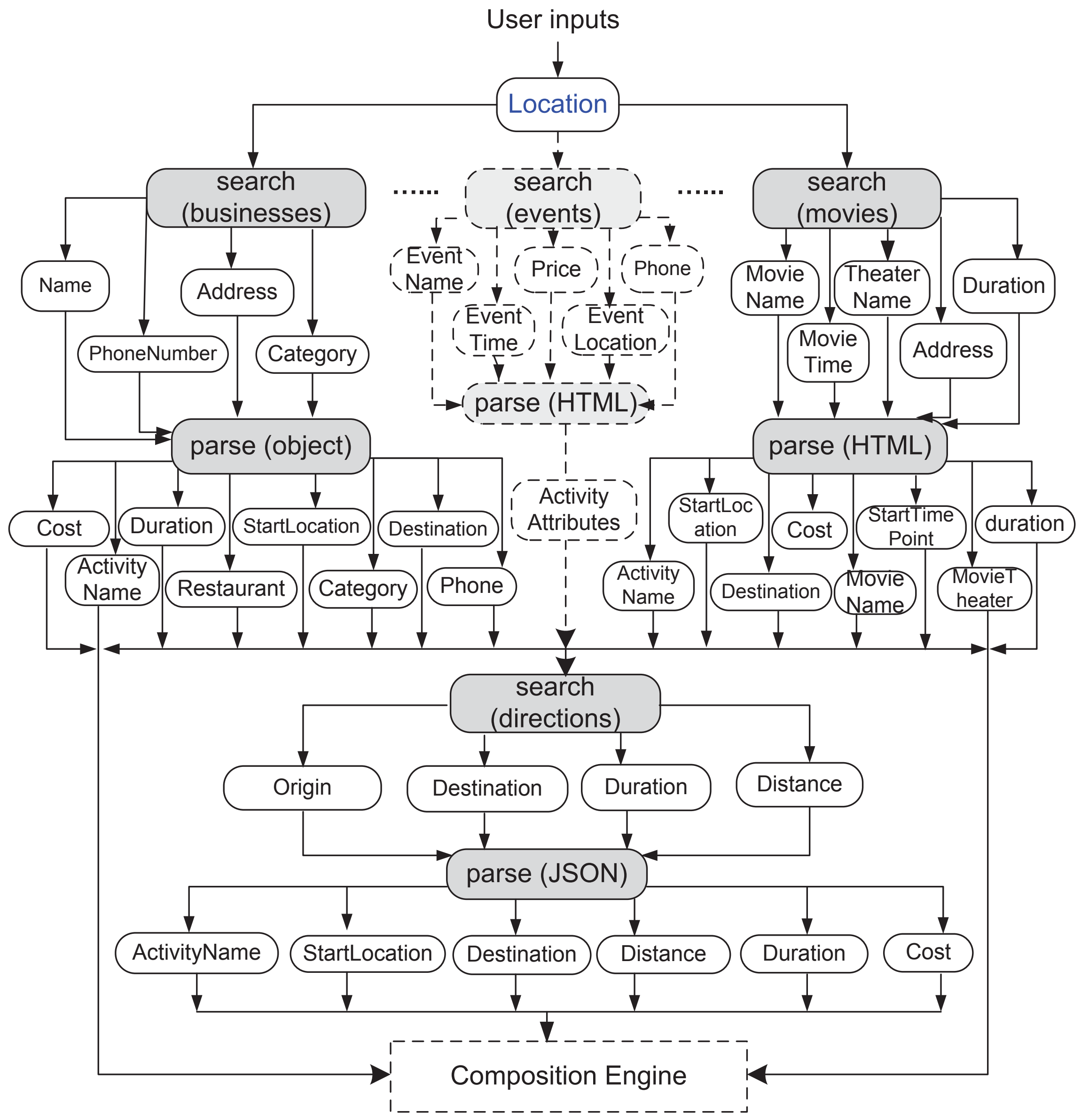


Figure 11: The data flow of the mashup process

Chapter 5

Service Composition

Service composition is to generate a business process to fulfill a functional goal that can not be achieved by individual services. After service discovery mashup, the outputs of different types of services are uniformed into the same format. Taking advantage of service mashup, service composition is performed on a HTML service, a SOAP service and a RESTful service provided with the outputs in an uniform format. The objective of service composition is to develop an entertainment planner over smart phones that can design some good entertainment plans according to users' requirements and recommend to users.

Service composition can be modeled as an AI planning problem. During the planning phase, a search tree to represent the problem space is constructed. Each state node in the search tree corresponds to a context of the problem space. Since our constraint context model defines constraints over propositions and real values, state nodes in the search tree also contain constraints over propositions and real values. Therefore, our search tree can handle both propositions and real values. In the following, we first define the symbols used in our service composition algorithm. In terms of the symbols, we then build the theoretical framework of the planning problem for our service composition.

5.1 Problem Description

A **system state** s represents a current context of the system.

Definition 7 *A **system state** s is a context, i.e. a set of constraints about variables.*

For example, one possible initial state s_0 for the motivating example in Section 2.5 can be described as below:

$$\begin{aligned} s_0 = \ & \{movie = false, dining = false, driving = false, \\ & t = 19:00, l = \text{“}H3K2S5\text{”}, m = 100\} \end{aligned} \tag{3}$$

In the initial context as shown in Equation 3, the user does not have any entertainment activities, such as watching some movies, having a dinner or driving out. He stays at a place that can be identified by the postcode “H3K2S5” at 19 : 00. His budget for all the entertainments is 100 dollars.

Definition 8 *A* ***service composition query*** *is a tuple* $\langle s_g, s_0, C\rangle$*, where*

- s_g *is a goal state;*
- s_0 *is an initial state.*
- C *is a set of constraints at any time.*

For a service composition query, s_g contains both a set of expected business goals that users want to achieve and a set of the constraints on the duration, location and cost, s_0 is the initial state (an example can be find in Equation 3), and C is the constraints for any state. A possible composition query q for the motivating example in Section 2.5 could be as follows:

$$\begin{aligned} q = \ & \langle s_g, s_0, C\rangle, \text{ where} \\ & s_g = \{movie = true, dining = true, \\ & \qquad t_g - 19:00 \le 240, l_g = \text{“}H3K2S5\text{”}, m_g \ge 0\}, \\ & s_0 = \{movie = false, dining = false, driving = false, \\ & \qquad t_0 = 19:00, l_0 = \text{“}H3K2S5\text{”}, m = 100\}, \\ & C = \{|l - \text{“}H3K2S5\text{”}| \le 20000\} \end{aligned} \tag{4}$$

In Equation 4, s_g specifies the expected business goals which require the user to see the movie and have dinner to entertain himself. s_g also require that the user spends no more than 100 dollars in 4 hours (240 minutes) starting from 19:00. The initial state s_0 is the same as in Equation 3. C means the user goes to any entertainment place that cannot be far more than 20 kilometers from his original starting place identified by the postcode “H3K2S5”.

Definition 9 *The **state transition function** γ of $a = \langle Pre_a, Attr_a, P_a^+, P_a^- \rangle$ for any state s is $\gamma(s, a) = s'$, iff a is applicable to s, i.e. $s \succ a$.*

The state transition function of an activity defines the conditions that the activity a can be executed at a state s and the effects after it is executed. The example of a state transition function γ of the activity a for the state s_0 described in the Equation 3 is $\gamma(s_0, a) = s'$. Suppose the dining activity denoted in Equation 1 in Section 3.1 can be applied to s_0, then a new state s' is generated by the transition $\gamma(s_0, a)$ as follows:

$$\begin{aligned} \gamma(s_0, a) = \ & \{movie = false, dining = true, driving = false, t = 21:00, \\ & l = \text{“}2501 NotreDame, Montreal, QC, CA, H3J1N6\text{”}, m = 75\} \end{aligned} \tag{5}$$

According to Equation 5, the business purpose of dining is achieved because $dining = true$ and the user has 75 dollars left in state s'. The time after dinner is $21:00$ o'clock.

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

Definition 10 *A **service composition problem** is a tuple $\langle s_g, A, s_0, C \rangle$, where*

- *s_g is a goal state;*
- *A is a set of available activities;*
- *s_0 is the initial state;*
- *C is a set of constraints satisfied at any time.*

For a set of available activities A, a service composition problem is to generate a business process that can both achieve all the business goals and satisfy the constraints in s_g from the initial state s_0 provided that all the constraints in C are satisfied during the composition process. The constraints in C are specified over real value variables, *e.g.* the duration constraint, the location constraint, and the cost constraint.

For example, a composition problem can consist of s_0 denoted in Equation 3, s_g denoted in Equation 4, and $A = \{a_1, a_2, ..., a_n\}$. For any state s_i, C can be expressed as follows:

$$\begin{aligned} C = \ & \{t_i - 19:00 \leq 240, \\ & |l_i - \text{“}H3K2S5\text{”}| \leq 20000, \\ & m_i \leq 100\} \end{aligned} \tag{6}$$

In Equation 6, $t_i - 19:00 \leq 240$ is a duration constraints, $|l_i - \text{“}H3K2S5\text{”}| \leq 20000$ is the location constraint, and $m_i \leq 100$ is the cost constraint.

In the motivating example in Section 2.5, A is a series of entertainment activities about watching movies, having dinners and driving. According to Definition 4, a_1 and a_2 are different activities if they have different preconditions, positive effects, negative effects or constraints. Even if both a_1 and a_2 are activities about watching the same movie, a_1 is different from a_2 if the movie theater or show time of a_1 is different from that of a_2.

Definition 11 *A **solution** π to the service composition problem $\langle s_g, A, s_0, C\rangle$ is a sequence of activities $\langle \pi_1; \ldots; \pi_n\rangle$, in which each π_i ($i \in [1, n]$) is a set of paralleled activities. π_1 is applicable to s_0. π_i is applicable to $\gamma(s_{i-2}, \pi_{i-1})$ when $i = [2, n]$. s_g hold at a goal state $s_g = \gamma(\ldots(\gamma(\gamma(s_0, \pi_1), \pi_2)\ldots\pi_n)$. C are satisfied at any state s_i, $i \in [0, n]$.*

For example, the ***Option 1*** for the motivating example in Section 2.5 can be represented as $\pi_1 = \{a_1, a_2\}$, where a_1 represents the activity of having dinner at Restaurant L'Autre Saison and a_2 represents the activity of watching movie “The Help”at cinema “Cinema Banque Scotia Montreal”.

5.2 Service Composition Algorithm

We use beam search algorithm to solve service composition problem. Beam search algorithm uses breadth-first search to build its search tree and can handle different types of constraints easily. At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost. However, it only stores a predetermined number of states at each level (called the beam width). The greater the beam width, the fewer states are pruned. With an infinite beam width, no states are pruned and beam search is identical to breadth-first search. As far as we know, beam search is used for the first time to solve service composition by this thesis. It is chosen because we can easily control the computation resource it uses by adjusting the beam width, which is very important for mobile devices. Since a goal state could potentially be pruned, beam search sacrifices completeness (the guarantee that an algorithm will terminate with a solution, if one exists) and optimality (the guarantee that it will find the best solution).

Algorithm 1 Beam search algorithm

```
1: $Q.insert(s_0)$ and mark $s_0$ as visited;
2: while $Q \neq 0 \vee$ number of solutions $< k$ do
3:     $s \leftarrow Q.popFirst()$;
4:     if $s_g$ is satisfied at $s$ then
5:         retrieve solution;
6:     for $\forall a$ applicable to $s \wedge C$ are satisfied after applying $a$ do
7:         $s' = \gamma(a, s)$;
8:         if $s'$ not visited then
9:             mark $s'$ as visited, remember $s'$'s parent as $s$;
10:            $Qtemp.insert(s')$;
11:        else
12:            prune the branch from the duplicated node to $s'$;
13:    if $Q = 0$ then
14:        sort $Qtemp$ with heuristic cost, keep only $k$ top elements;
15:        move all elements from $Qtemp$ to $Q$;
```

Algorithm 1 presents the detailed steps of beam search algorithm. Beam search algorithm starts from the initial state s_0. Line 1 marks s_0 as visited and inserts s_0 into queue Q. While Q is not empty and the obtained solutions are less than k (line 2), a loop starts. Q pops out a state from the head of the queue and names the state as the current state s (line 3). If all the constraints in the goal context s_g are satisfied at the current state s (line 4), a solution is retrieved from the tree (line 5). For any activity a is applicable to s and constraint C is satisfied after applying a (line 6), a new state s' is generated from the state transition $\gamma(a, s)$ (line 7). If s' is not visited (line 8), we mark s' as visited and record s as the parent of s' (line 9). s' is also inserted into the queue *Qtemp* (line 10). Otherwise, the whole branch rooted at s' is pruned from the tree (line 12). If Q is empty (line 13), line 14 sorts the nodes in *Qtemp* according to the heuristic cost and keeps only top k nodes. All the element of *Qtemp* are moved into Q (line 15). Then, the next iteration starts.

The heuristics in our application is that we prefer the activities that cost less money and closer to the original location, in addition, the location should be convenient for finding more entertainment activities. Therefore, we use the following heuristic function (Eq. 7) to compare a set of states $s_1, \cdots, s_n$. In Equation 7, $m_{max} = max(\{s_i.m\})$ and $m_{min} = min(\{s_i.m\})$ are the maximum and the minimum money a state can have. $D0_{max} = max(\{|s.l - l_0|\})$ and $D0_{min} = min(\{|s.l - l_0|\})$ are the maximum and the minimum distance to the original location l_0. $N_{max} =$

$max(s_i.activities)$ and $N_{min} = min(s_i.activities)$ are the maximum and the minimum number of available activities a state can have.

$$h_s = \frac{s.m - m_{min}}{m_{max} - m_{min}} \times 0.6 + \frac{D0_{max} - |s0.l - s.l|}{D0_{max} - D0_{min}} \times 0.2 + \frac{N_{max} - s.activities}{N_{max} - N_{min}} \times 0.2 \tag{7}$$

Algorithm 1 terminates after k solutions are reported or no states to expand.

Theorem 1 *The time complexity of beam search algorithm is linear to the beam width k and the maximum depth m.*

Proof 1 *k is the beam width and m is the maximum depth of any path in the search tree. The beam search algorithm only expand at most k nodes at each level. Thus, the worst case time is $O(km)$.*

Chapter 6

Implementation of the Entertainment Planner

In this chapter, we present the design and the implementation of the entertainment planner for smart phones with the Google Android operating system.

6.1 Introduction

6.1.1 Goals and objectives

The goal of this system is to produce an interactive and entertainment application for the users who expect entertainment plans for a period of time. This application is playable on any phone supporting the Android operating system version 2.3 or above with access to the Internet.

6.1.2 Statement of scope

The inputs to this application are the user's selection and input, and the location detected by the GPS module of the phone, and the only output is the entertainment planners that is a consequence of the input.

6.1.3 Major constraints

In order for the entertainment planner application to be installed the user must be using a mobile device running Google's Android Operating System version 2.3 and above along with a GPS module, as well as having access to the Internet.

The application resides a Android operation system version 2.3 or above. A JSON formatted database resides on the system to store the information from the service mashup phase.

6.2 Design Consideration

6.2.1 Constraints

All development for the entertainment planner is done in the Eclipse Integrated Development Environment (IDE) on Windows 7 machines with the Android Software Development Kit (SDK). Testing of the application was done on the HTC Incredible S and HTC Wildfire S. As long as the Internet access and a GPS module are available to Google's Android mobile Operating System users, the application will be maintainable and functional.

6.2.2 System Environment

We test our application in the following two kinds of smart phones.

1. • Platform: HTC Incredible S
   • Operation System: Google's Android Operating System version 2.3.4
   • Hardware:
     – Processor: 1GHz
     – Memory: 768MB
     – Screen Size: 480 × 800 pixels, 4.0 inches
     – microSD: 8GB
     – GPS: A-GPS support
2. • Platform: HTC Wildfire S
   • Operation System: Google's Android Operating System version 2.3.3
   • Hardware:
     – Processor: 600 MHz

- Memory(RAM): 512 MB
- Screen Size: 320 × 480 pixels, 3.2 inches
- microSD: 2GB
- GPS: A-GPS support

6.3 System Architecture

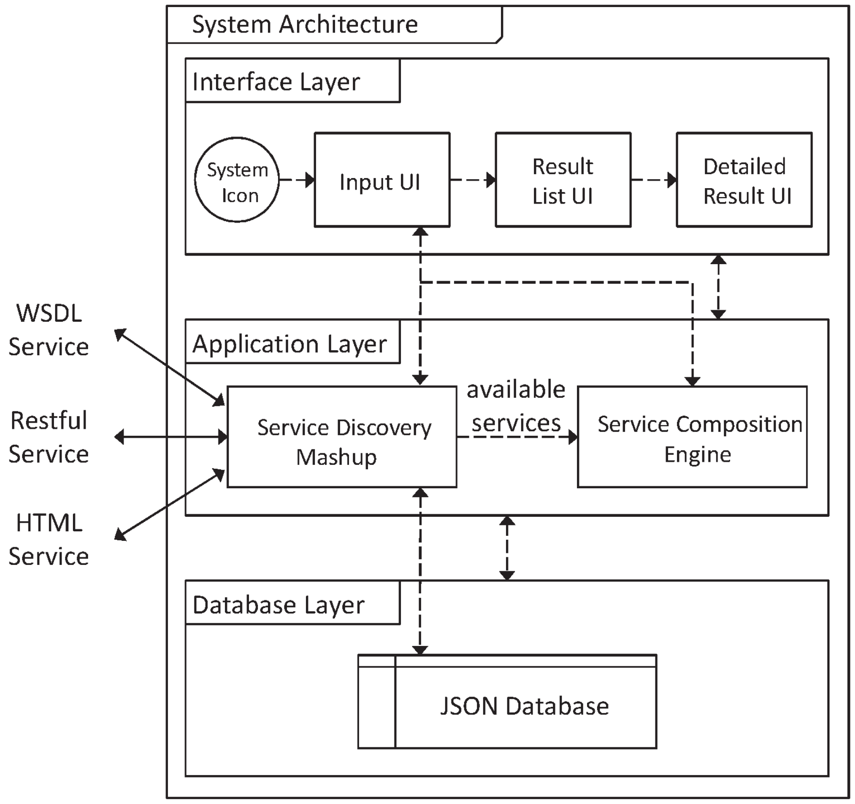


Figure 12: System Architecture Diagram

As shown in Figure 12, the system follows the three-tier architectural style and is organized into three layers: the interface layer, the application layer and the storage layer. The interface layer is the graphical user interface that allows the users to interact with the system. The application layer contains two functionalities: Web service discovery mashup and composition. It mashups and composes Web services, and parses and stores data. It contains the logic and rules for storing data in the database layer and also retrieving it in accordance with the user's needs. This is the layer that contains the data file parsers and allows controlled access to the data files in the mashup phase. Finally, the storage layer stores the JSON formatted metadata required for the system.

The three-tier architecture style is used because it not only separates the user interface and the metadata, but also provides an application logic layer. The application layer provides a middle layer that allows the data files and the UI components to be loosely coupled. The application layer has to be modified if there are any changes to the format of the data files and the interface layer will need little or no modification. This makes it easy for clients of this application to modify the data file format and attributes for further research purposes if they wish to do so. This layer makes the system more maintainable and reusable and also hides the complexity of processing data from the users.

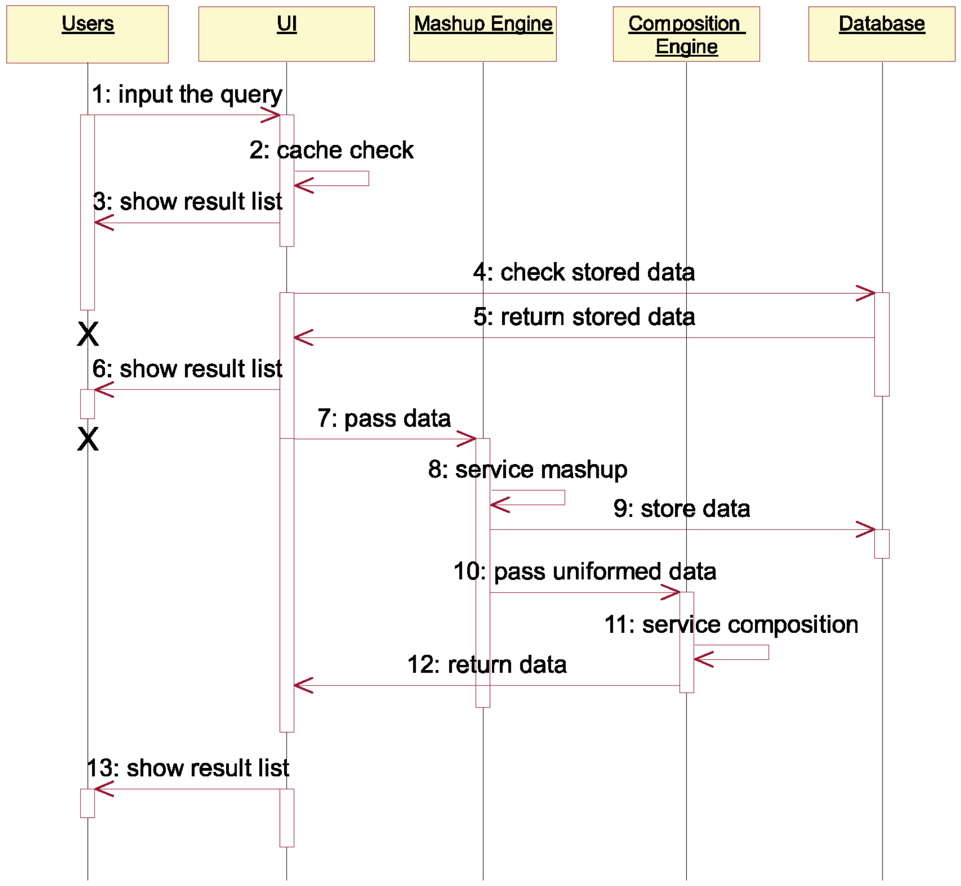


Figure 13: System Sequence Diagram

Smart phone users interact with the entertainment system through a visible UI on the smart phone. Users click the system icon to start the application. As the system sequence diagram shown in Figure 13, when the entertainment planner is launched, the user is presented with the *Input UI* for him to input his query. Once a query from the client is acquired, the system directly returns the cached data to the user if such data exists. Otherwise, the interface layer invokes the database layer to check if there exists stored data, it returns the stored data if it exists. If there does not exist cached data or stored data, the application layer invokes the mashup engine. It invokes three different types of services, *i.e.* HTML services, SOAP services and RESTful services to generate all data as the required uniform format and store them into the storage layer. Then, the step of service mashup parses the data stored as the input of service composition step. After parsing is done, the planner generates plans using AI planning based service composition algorithm. Once the result is ready, the entertainment planner displays a summary list of the most ideal top 10 plans for the user in the *Result List UI*. The user is able to see the details of any plan in the summary list by clicking the the plan item, which is displayed in the *Detailed Result UI*.

In accordance to the system functionality, we split our entertainment planner system as three functional modules: User Interface and Functions, Service Discovery Mashup Implementation, and Service Composition Implementation. We present them in detailed in the following sections.

6.4 User Interface and Functions

6.4.1 Design constraints

The design of the interfaces for the entertainment planner is simple and intuitive so that the user can easily identify what options they currently have to progress. Each of UIs responds quickly to users input through the buttons on the screen. When any operation needs more time to response, the progress bar is shown to avoid frustrate the user or lead to believe that the application is frozen.

6.4.2 System Icon

Figure 14 shows available application page displayed on the smart phone. Our entertainment planner application is represented by a red frame icon. A smart phone can start the application by clicking on the icon. Screen shots are show using HTC

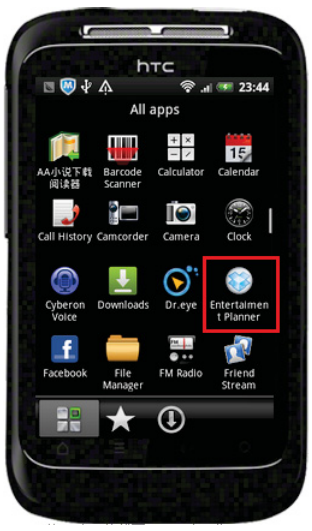
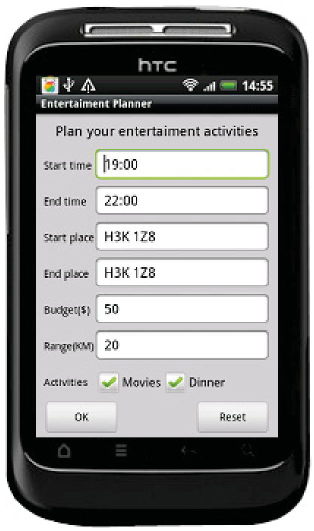


Figure 14: System icon


htc 14:55
Entertaiment Planner

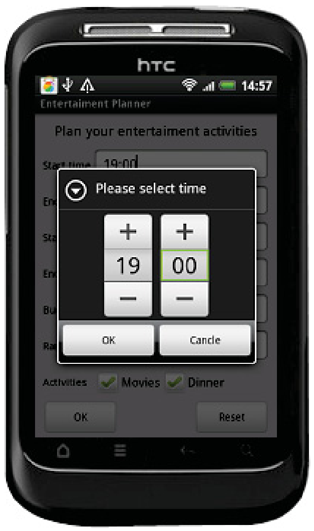
Plan your entertaiment activities

Start time	19:00
End time	22:00
Start place	H3K 1Z8
End place	H3K 1Z8
Budget($)	50
Range(KM)	20

Activities ✓ Movies ✓ Dinner

OK Reset

Figure 15: Input UI


htc 14:57
Entertaiment Planner

Plan your entertaiment activities

Please select time

19 00

OK Cancle

Activities ✓ Movies ✓ Dinner

OK Reset

Figure 16: Setting time

Wildfire S.

6.4.3 Input UI

We design an input UI as shown in Figure 15. For example, a user is on a business trip in Montreal. In order to ease the step of filling in users' queries, the default setup is displayed in the initial UI which are automatically entered. The default start time and end time are 19:00 and 23:00 respectively. The start place and the end place are detected by our application as where the user currently is, so that users do not need to give the exact address of their current location. By default the budget for all entertainments is no more than 50 dollars. If the user wants to make his own plan, he can edit all the fields in the Input UI as shown in Figure 15. Suppose the user follow the default setup as shown in Figure 15, he plans to watch movies and have dinner somewhere within 20 kilometers during the time, and expect our entertainment application provide them with 10 ideal plans.

When the user inputs the start time or the end time, the application will pop up a dialog as shown in Figure 16 for the user to select the time.

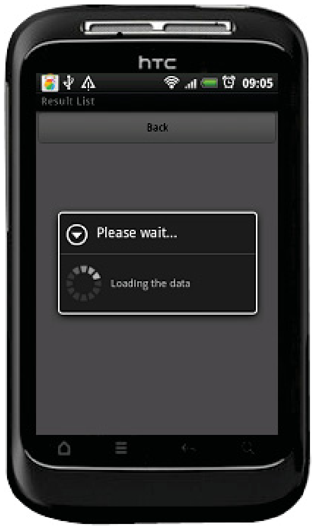
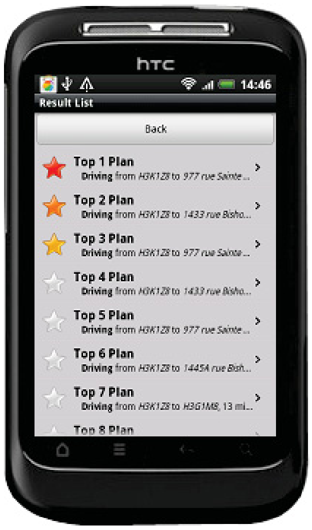


Figure 17: Progress



Top 1 Plan
Driving from *H3K1Z8* to *977 rue Sainte* ...

Top 2 Plan
Driving from *H3K1Z8* to *1433 rue Bisho*...

Top 3 Plan
Driving from *H3K1Z8* to *977 rue Sainte* ...

Top 4 Plan
Driving from *H3K1Z8* to *1433 rue Bisho*...

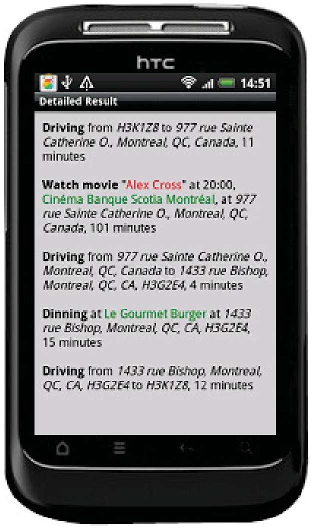
Top 5 Plan
Driving from *H3K1Z8* to *977 rue Sainte* ...

Top 6 Plan
Driving from *H3K1Z8* to *1445A rue Bish*...

Top 7 Plan
Driving from *H3K1Z8* to *H3G1M8*, 13 mi...

Top 8 Plan

Figure 18: Result List UI



Driving from *H3K1Z8* to *977 rue Sainte Catherine O., Montreal, QC, Canada*, 11 minutes

Watch movie "Alex Cross" at 20:00, Cinéma Banque Scotia Montréal, at *977 rue Sainte Catherine O., Montreal, QC, Canada*, 101 minutes

Driving from *977 rue Sainte Catherine O., Montreal, QC, Canada* to *1433 rue Bishop, Montreal, QC, CA, H3G2E4*, 4 minutes

Dinning at Le Gourmet Burger at *1433 rue Bishop, Montreal, QC, CA, H3G2E4*, 15 minutes

Driving from *1433 rue Bishop, Montreal, QC, CA, H3G2E4* to *H3K1Z8*, 12 minutes

Figure 19: Detailed Result UI

6.4.4 Result List UI and Detailed Result UI

When the user query is submitted, the service mashup and composition are executed. Since the execution of service mashup and composition needs some time, a progress bar shows to the user (see Figure 17). One the data is ready, the plans are returned and the result list UI is displayed to the user with the summary list. As shown in Figure 18, the entertainment planner generate top 10 plans for the user to spend 4 hours doing entertainment activities after 7:00 p.m. The user clicks any plan in the list to see the details of the plan. For example, a good plan for a user as in Figure 19 is presented in the Table 13.

Table 13: A good plan

Seq	Activity	Place	Time
1	Driving	from H3K2S5 to 2501 Notre Dame, Montreal, QC, H3J1N6	5 minutes
2	Dining	in the restaurant “Liverpool House” at 2501 Notre Dame, Montreal, QC, H3J1N6	30 minutes
3	Driving	from 2501 Notre Dame, Montreal, QC, H3J1N6 to 2313 St. Catherine St. West Suite 101, Montreal, QC, Canada	6 minutes
4	Watching Movie	in the theater “Your Sister’s Sister” AMC Forum 22 at 2313 St. Catherine St. West Suite 101, Montreal, QC, Canada	90 minutes from 19:50

6.5 Service Discovery Mashup Implementation

Since Google Android SDK is Java based language, we would like to take its benefits by modeling mashup problem in an object-oriented way. As Figure 20 shows, we model service discovery mashup (see Section 4.6) using the following objects.

- `ServiceParser(services)` At the top, there is an interface named `ServicePa-rser` in the Java package `services`, which defines an abstract method named `generateActivityFiles` that can be implemented by other classes to return an activity list, and at the meantime generate JSON files to store those activities for caching. `MovieParser`, `DiningParser` and `DrivingParser` implement `ServiceParser`, generate the files in the JSON format, and return a List result containing a kind of relative `Activity`.

- `Activity(algorithm)` is an abstract class in the Java package `algorithm` defined in the Definition 4 in Chapter 3.

- `Movie`, `Dining`, and `Driving` extend `Activity` class and represent a specific kind of activity respectively.

- `MovieParser` invokes an HTML service and parses the result into a `Movie` activity list.

- `DiningParser` invokes a SOAP service and parses the result into a `Dining` activity list.

- `DrivingParser` invokes a RESTful service and parses the result into a `Driving` activity list.

6.5.1 Ontology Data Mapping

In order to make use of the ontology defined in Chapter 3, we map the ontology in Table 2 to class *Ontology*, which is *Enum* type, by taking advantage of the object-oriented method. By constructing an instance of class *Ontology*, all the concepts defined in Table 2 become available in our program. During the procedure of mashup, all the tag values in the JSON files can be obtained from class *Ontology*. The following code segments representing activities show how Ontology is applied in our programming.

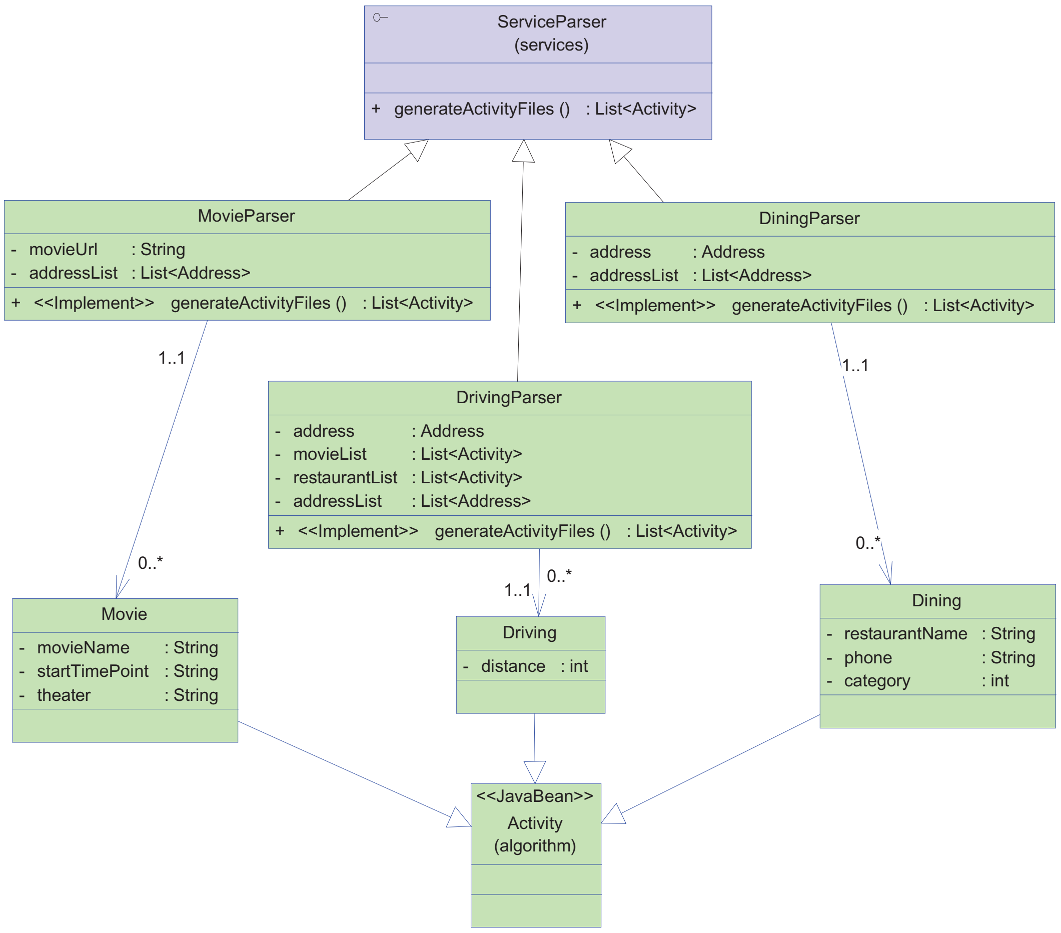

ServiceParser
(services)
+ generateActivityFiles () : List<Activity>
MovieParser
- movieUrl : String
- addressList : List<Address>
+ <<Implement>> generateActivityFiles () : List<Activity>
DiningParser
- address : Address
- addressList : List<Address>
+ <<Implement>> generateActivityFiles () : List<Activity>
DrivingParser
- address : Address
- movieList : List<Activity>
- restaurantList : List<Activity>
- addressList : List<Address>
+ <<Implement>> generateActivityFiles () : List<Activity>
1..1
0..*
1..1
0..*
1..1
0..*
Movie
- movieName : String
- startTimePoint : String
- theater : String
Driving
- distance : int
Dining
- restaurantName : String
- phone : String
- category : int
<<JavaBean>>
Activity
(algorithm)

Figure 20: Service Mashup Class Diagram

- Movie Activity

  The attributes of each movie activity include *ActivityName*, *StartLocation*, *Destination*, *Cost*, *Duration*, *MovieName*, *StartTimePoint* and *MovieTheater*. *StartLocation* and *Destination* are the same location because a movie show starts and ends at the same place. The attribute *MovieName* is to avoid watching the same movie in one plan during composition procedure.

```
{
    "type":{"id":"52","ActivityName":"Movie"},
    "StartLocation":"350 Rue Emery, Montreal, QC, Canada,",
    "Destination":"350 Rue Emery, Montreal, QC, Canada,",
    "Cost":"13.0",
    "Duration":"88",
    "MovieName":"Attack the Block",
    "StartTimePoint":"19:35",
    "MovieTheater":"Cinema Banque Scotia Montreal"
},
```

- Dining Activity

  The attributes of each dining activity include *ActivityName*, *StartLocation*, *Destination*, *Cost*, *Duration*, *Category*, *Restaurant* and *Phone*. *StartLocation* and *Destination* are the same location because a dining activity starts and ends at the same *Restaurant*.

```
{
    "type":{"id":"262","ActivityName":"Dining"},
    "StartLocation":"2501 Notre Dame, Montreal, QC, CA, H3J1N6",
    "Destination":"2501 Notre Dame, Montreal, QC, CA, H3J1N6",
    "Cost":"25.0",
    "Duration":"120",
    "Restaurant":"Liverpool House",
    "Phone":"(514) 313-6049",
},
```

- Driving Activity

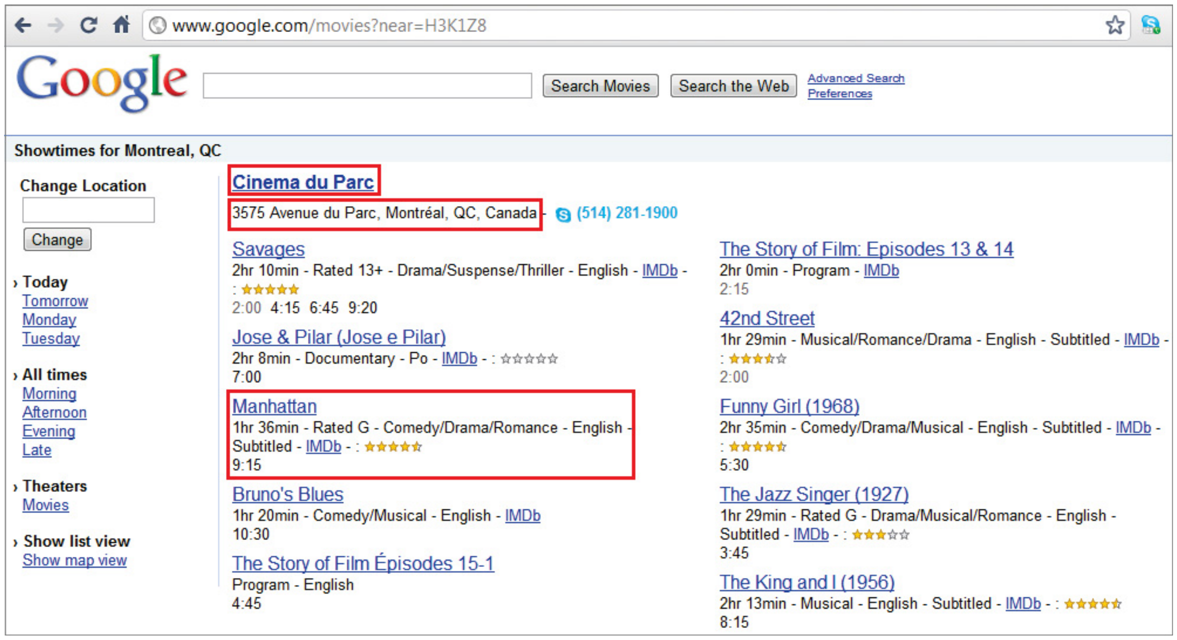
  The attributes of each driving activity include *ActivityName*, *StartLocation*, *Destination*, *Cost*, *Duration* and *Distance*. Normally, *StartLocation* and *Destination* are different locations. Because a driving activity always drives a person from one place to another place except if the person returned to his *StartLocation*. No matter if *StartLocation* and *Destination* are the same or not, a distance always exists if a driving activity is applied.

```
{
    "type":{"id":"275","ActivityName":"Driving"},
    "StartLocation":"H3K1Z8",
    "Destination":"3575 Avenue du Parc, Montreal, QC, Canada,",
    "Cost":"19.0",
    "Duration":"14",
    "Distance":"5055",
},
```

6.5.2 HTML Service

For the HTML service, we use the Google Show Time website (see Section 4.5) `http://www.google.com/movies?near=Location`, where *Location* is a postal code, a city name or a detailed address. For example, we get the following HTML page, as shown in Figure 21 for the url `http://www.google.com/movies?near=H3K1Z8`. There are many information about movies presented in the HTML page in Figure 21.

For example, "Cinema du Parc"is the name of the movie theater, under which the address and the telephone number of the movie theater are also displayed. A list of detailed movie information is presented after the information of the movie theater. The detailed information about movie includes the name, duration and start time of a movie. As can be seen from Figure 21, the movie called "Manhattan"starts at "9:15"and the duration is "1hr36min". It is possible that one movie will be shown several times a day. For example, the movie called "Savages"starts at 2:00, 4:15, 6:45 and 9:20. In our implementation, we transform movies into movie activities and consider movies, even if with the same name, but with different show time or shown in different theater as different movie activities.



Showtimes for Montreal, QC

Cinema du Parc
3575 Avenue du Parc, Montréal, QC, Canada - (514) 281-1900

Savages
2hr 10min - Rated 13+ - Drama/Suspense/Thriller - English - IMDb - : ★★★★★
2:00 4:15 6:45 9:20

Jose & Pilar (Jose e Pilar)
2hr 8min - Documentary - Po - IMDb - : ☆☆☆☆☆
7:00

Manhattan
1hr 36min - Rated G - Comedy/Drama/Romance - English - Subtitled - IMDb - : ★★★★★
9:15

Bruno's Blues
1hr 20min - Comedy/Musical - English - IMDb
10:30

The Story of Film Épisodes 15-1
Program - English
4:45

The Story of Film: Episodes 13 & 14
2hr 0min - Program - IMDb
2:15

42nd Street
1hr 29min - Musical/Romance/Drama - English - Subtitled - IMDb - : ★★★★★
2:00

Funny Girl (1968)
2hr 35min - Comedy/Drama/Musical - English - Subtitled - IMDb - : ★★★★★
5:30

The Jazz Singer (1927)
1hr 29min - Rated G - Drama/Musical/Romance - English - Subtitled - IMDb - : ★★★☆☆
3:45

The King and I (1956)
2hr 13min - Musical - English - Subtitled - IMDb - : ★★★★★
8:15

Figure 21: Showtimes Screen shot

We use jsoup library to invoke and parse HTML into the JSON document (JSON segment can be seen in Section 6.5.1) to store the movie activity list. Jsoup is a Java based light-weight API for working with HTML. It provides very convenient methods for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods [Hed12].

With the specified HTML, *e.g.* the HTML from Google Show Time as shown in Section 6.5.2, to locate `<div id="movie_results">` we use the following code:

`doc.select("div[id=movie_results]")`, where `doc` is a instance of `Document` in jsoup library.

Similarly, we can get the theater name "IMAX TELUS Montreal Science Centre" in HTML shown in Section 6.5.2 by the following code:

`doc.select("div[class=theater] > h2[class=name]")`, where `doc` is the same as above.

6.5.3 SOAP Service

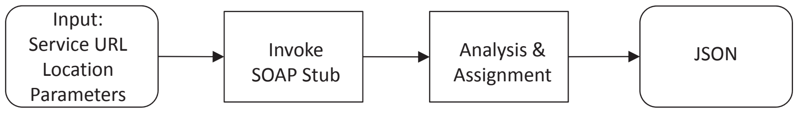


Figure 22: Soap Service

In order to be able to invoke Bing Maps SOAP Service (see Section 4.3), we first need to create a Bing Maps account to get a Bing Maps key. The key is set up in `SearchRequest.Credentials` to identify the requester. We also need to set up the properties of the request, *e.g.* `ListingType=Business`, `Filters.Category=11168` and etc. The procedure to invoke Bing Maps SOAP service is shown in Figure 22.

Bing Maps SOAP service has four services as described in section 4.3. We utilize search service of them. Bing Maps SOAP service is from Microsoft and written by .Net, which is very unfriendly to Java based Android library. In order to use it, we have to remove all .Net policies from its WSDL file and wrap it with our own Java SOAP service. Then, we deploy our SOAP service onto the Amazon Elastic Compute Cloud (Amazon EC2) [ama12] such that our SOAP service can be invoked publicly. The WSDL can be found at `http://ec2-107-21-247-191.compute-1.amazonaws.c-om:8080/bingsearch/BingMapSearchPort?wsdl`. In our implementation, we use 3rd party library ksoap2-android [mos11] to invoke the wrapped SOAP service since Android does not come with SOAP library.

The process of "Analysis and Assignment" in Figure 22 is necessary. Because we are not able to obtain all the information about the restaurant from the SOAP service, *i.e.* cost and duration. We have to specify the cost and duration for each restaurant by iterating the result list and assign some ideal values for them in according to the category of restaurants as shown in Table 14. A dining activity in final JSON format can be seen in Section 6.5.1.

Table 14: Cost and Duration for Restaurants

Category of restaurants	cost	duration
Restaurants	\$25	120 minutes
Pizza	\$15	20 minutes
Bars	\$30	180 minutes
Grocery & Food Stores	\$10	10 minutes
Others	\$15	15 minutes

6.5.4 RESTful Service

We use the Google map RESTful service (see Section 4.4) to get direction between two locations. The procedure to invoke the RESTful service is shown in Figure 23. Suppose we have m restaurants and n movies. We need to invoke RESTful service $m \times n$ times to get the driving direction among the user's place, restaurants and movie theaters. Since we only own free license, there is a limitation to invoke it per day (2,500) [Goo12b]. In our algorithm, we have to get all directions between any two different types of locations, *e.g.* the direction between a movie theatre and a restaurant, it means the locations cannot be more than 50. Actually, in order to improve the performance over smart phone, we only get the nearest 10 movie theaters and 10 restaurants around the user's current location. A driving activity in JSON format can be seen in Section 6.5.1.

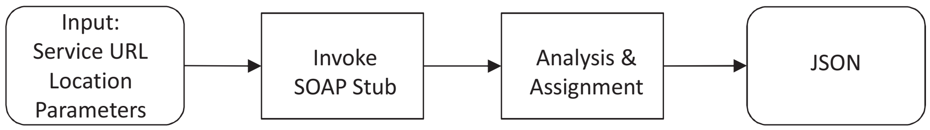


Figure 23: RESTful Service

6.6 Service Composition Implementation

6.6.1 Building Models

To implement service composition, we model service composition using nine objects, as shown in Figure 24. Web service composition problem can be mapped into an AI planning problem and beam search is used as our planning algorithm in our work. Taking advantage of the object-oriented method, we define the following objects in the implementation of the beam search algorithm using Java language.

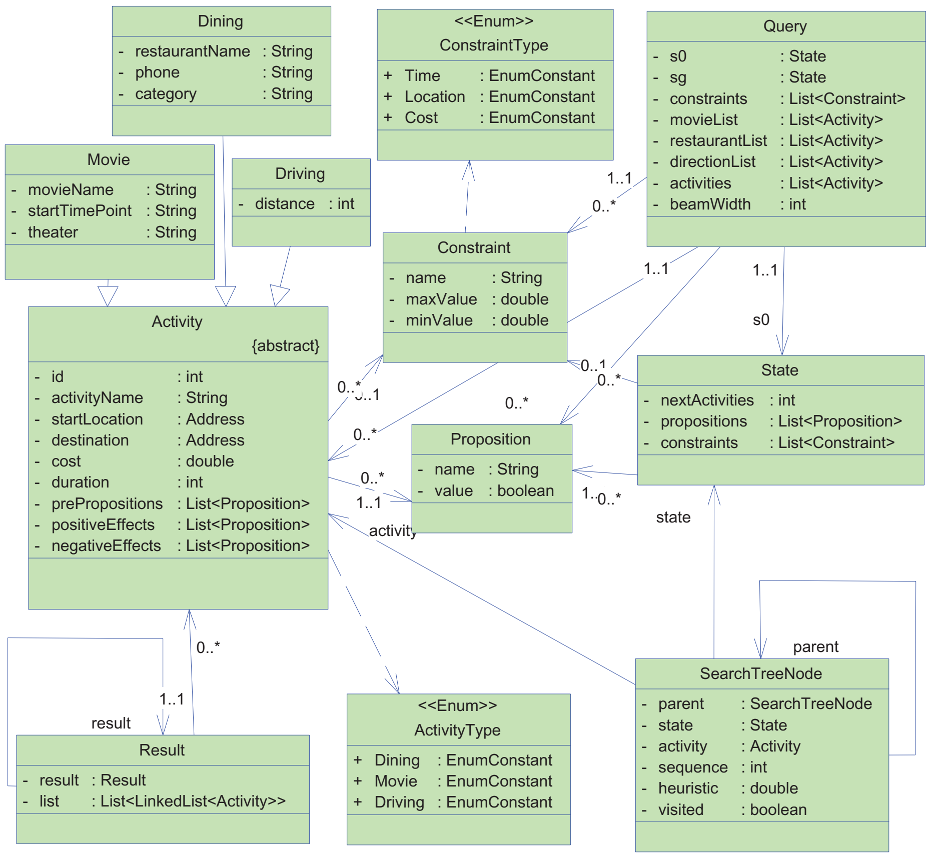


Figure 24: Class Digram for Models

- `ConstraintType`

  `ConstraintType` is an object of enumerate type, which makes a constraint on the property `name` of class `Constraint`. In other words, the value of the property `name` of `Constraint` comes from the instances of `ConstraintType`, *i.e.* `Time`, `Location`, and `Cost`. In our algorithm, to improve the performance we finally do not use this class instead using corresponds string directly.

- `Constraint`

  To take advantage of object-oriented language, we seperate the constraints on real value variable and on proposition symbols to two classes. Here, `Constraint` object represents a constraint on real value. For example, our planner is required to find an entertainment plan to spend three hours nearby according to a user's inputs. Then, an instance of `Constraint` class is generated. The `name` is the string "Time" with the maximum value of 180 minutes and the minimum value of 0 minute.

- `Proposition`

  In our work, class `Proposition` represents the constraint on proposition symbols, it has two properties, *i.e.* `name` and `value`. The value of the property `value` can be *true* or *false*, which represents a status of the system. For example, `Movie=true` or `Movie=false` can be explained as whether the activity "Movie"has been done or not. In our program, `Proposition` is always defined as an element of a List, which is a property of another object to show its multiple statuses on different activities.

- `State`

  As described in Definition 7, a `State` object is a context, *i.e.* a set of constraints over real value variables and proposition symbols. The property `propositions` represents a List of `Proposition` objects, and property `constraints` represents a List of `Constraint` objects.

- `ActivityType`

  `ActivityType` is an abstract object of data type `Enum`, which is to restrict the property `ActivityName` of class `Activity`. It contains three instances: `Dining`, `Moving` and `Direction`. We do not use this class either for the same reason as `ConstraintType`.

- `Activity`

In according to the Definition 4, we define an `Activity` object that has the following properties, in which `activityName`, `startLocation`, `destination`, `cost`, and `duration` correspond the attributes of an activity.

  - `activityName`

    Property `activityName` denotes the current type of `Activity`, whose value is one of the following strings, *i.e.* "Dining", "Movie" or "Driving".

  - `startLocation` and `destination`

    Properties `startLocation` and `destination` are the starting place and destination of the current activity respectively. For the activities `Movie` and `Dining`, `startLocation` and `destination` are the same locations. But for `Driving` activity, `startLocation` and `destination` are two different locations.

  - `cost`

    Property `cost` is the cost of the current activity.

  - `duration`

    Property `duration` calculated by minutes is the time spent on the current activity.

  - `prePropositions`

    We use property `prePropositions` to represent a series constraints over proposition symbols, *i.e.* a `List` of `Propostion` objects. For example, an element of the `preConditions` of a `Dining` activity is `Dining=false`.

  - `positiveEffects` and `negativeEffects`

    Properties `positiveEffects` and `negativeEffects` are the positive effects and negative effects respectively added to a state after an activity is applied. For example, the positive effect and the negative effect of `Movie Activity` is `Movie=true` and `Movie=false`. After `Movie Activity` applies to a state, a new state is generated by removing `Movie=false` from the old state's propositions and then adding `Movie=true`. Therefore, the new state contains the proposition `Movie=true` and excludes `Movie=false`.

- `Movie`, `Dining`, and `Driving`

`Movie`, `Dining`, and `Driving` are sub-classes of class `Activity` to represent each specific type of `Activity`.

- `Query`

  As described in Definition 8, `Query` object is to describe a query sent by a user on the client side. Property `s0` and `sg` are initial state and goal state respectively. Property `constraints` is the requirement of the user, such as the total time the user want to spend, the distance range, and etc. Property `beamWidth` is the number of ideal plans that the user wants to get from the planner.

- `Solution`

  `Solution` object represents a plan returned by the planner as in Definition 11 , which encapsulates an instance of `LinkedList` data type, in which every element is an instance of `Activity`.

- `SearchTreeNode`

  Beam search algorithm builds the search tree in the way of breadth-first search. For each node in a search tree, we define a `SearchTreeNode`, which contains a variable of `SearchTreeNode` object to refer to its parent node. One instance of `SearchTreeNode` corresponds to one instance of `State`. Actually, a `SearchTreeNode` object encapsulates an object of `State` data type. Property `activity` is an activity can be applied to a state. Property `heuristic` is the value returned by heuristic function, *i.e.* heuristic cost. The higher the heuristic, the better the node is. By comparing the value of `heuristic`, we store nodes with higher heuristics in our search tree and excludes the rest. Property `visited` is to denote whether the node has been visited during the construction of the search tree.

6.6.2 Service composition Implementation

We implemented beam search algorithm (Algorithm 1) shown in Chapter 5. Here we utilize the Algorithm 1 as an example to illustrate our algorithm implementation. Suppose a user's inputs are as in Figure 15, the user expects to spend four hours from 7:00 p.m. near where he is. Currently, the user located in the area, identified as the postcode H3K2S5. The user does not want to go somewhere beyond 20 kilometers and will return to the current place. During the four hours, the user would like to spend no more than 50 dollars in watching movies and having dinner.

Since the beam search algorithm is AI planning based algorithm, we actually build a search tree with the implementation of the algorithm. When building the search tree, constraints play an important role to guide the building process. Constraint model is defined as context representation in our work. In fact, the search tree starting from the initial context is assumed to go to a series of current context and finally reach the goal context. In the following, we present how to build the search tree as well as the context change during the tree-building phase.

1. According to the inputs in the UI, we have the initial context and the goal context in Table 15. The initial context corresponds to state s_0. Starting algorithm from the initial state s_0. s_0 is inserted into the queue Q and marked as visited, where the values of attributes are from Table 15, *i.e.* $s_0 = \{mv_0 = false, dn_0 = false, dr_0 = false, t_0 = 19:00, l_0 = \text{“}H3K2S5\text{”}, m_0 = 100\}$.

2. While Q is not empty and the obtained solutions are less than k where $k = 10$, continue the loop:

3. Q Pops out a state from the head of the queue and names the state as the current state s, since now only s_0 in the queue, s_0 becomes the current state s. The search tree is as shown in Figure 25. And the initial context is the current context.

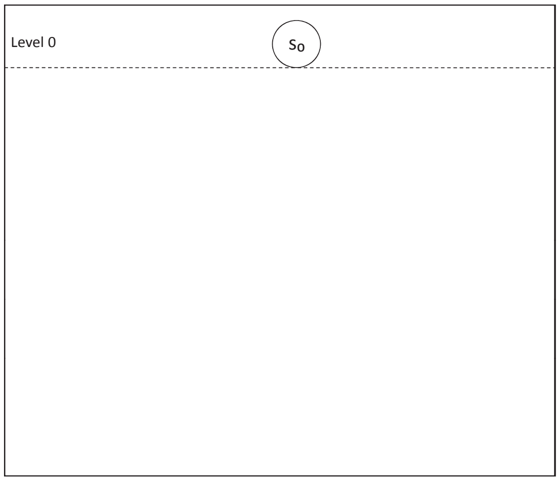


Figure 25: Search Tree Level 0

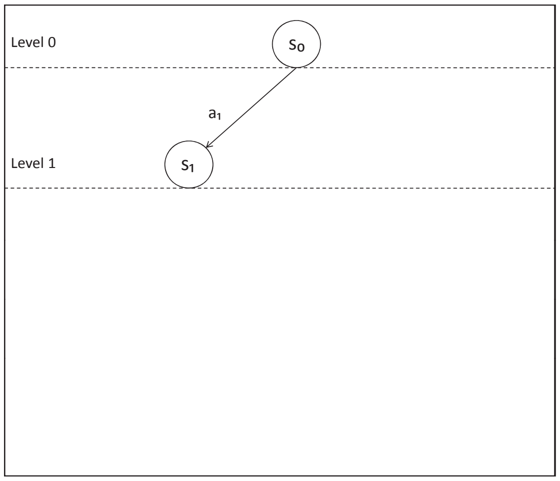


Figure 26: Search Tree Level 1.1

Table 15: Initial and goal context

variable Name	data type	ontology type	approximation operators	value	constraints
init-context					
l_0	string	start location		"H3K2S5"	$l_0 = \text{“}H3K2S5\text{”}$
m_0	real	possessed money		100	$m_0 = 100$
t_0	date	start time		19:00	$t_0 = 19:00$
mv_0	boolean	movie		false	$mv_0 = false$
dn_0	boolean	dining		false	$dn_0 = false$
dr_0	boolean	driving		false	$dr_0 = false$
goal-context					
l_g	string	destination		"H3K2S5"	$l_g = \text{“}H3K2S5\text{”}$
m_g	real	possessed money	greater than	0	$m_g \geq 0$
t_g	date	end time	near	19:00,4	$t_g - 19:00 \leq 4$
mv_g	boolean	movie		true	$mv_g = true$
dn_g	boolean	dining		true	$dn_g = true$
dr_g	boolean	driving	any of	true, false	$dr_g = true \vee dr_g = false$

4. If the goal state is satisfied at current state s, here we loop to compare if every proposition of goal all belongs to current state's propositions to make sure the check returns true.

5. A solution is retrieved from the tree. Because every node in the search tree stores its own parent node, we can get all its ancestral nodes, and all those nodes compose a path in a search tree, which is the solution we want to get.

6. For any activity a is applicable to s and constraint C is satisfied after applying a, in which we use a function named *isApplicable* to check if it's applicable or not. The function follows the steps:

   (a) First we check if the location of the state s is equal to the start location of the activity a, if so, do the next check, otherwise the activity a is not applicable to the state s;

   (b) Then we check if the preconditions over proposition symbols of activity a are included in the propositions of the state s;

   (c) We use s' here to express after applying a, and for checking if constraint C is satisfied s', we have to check the cost, the time, and the location range according to user's input.

7. In order to check those conditions in the steps 6, we look for an activity from the activity list that can be applied to the current state s_0. For example, we have a movie activity a_0 and the description of the movie activity in the JSON file is as the piece of code describing the movie activity shown in Section 6.5.1. The address of the movie theater is "350 Rue Emery, Montreal, QC, Canada". The movie starts at 19:35 and the duration is 88 minutes. The ticket is 13 dollars. We describe the context for this movie activity as shown in Table 16. Since the start location of activity a_0 is different from the location in state s_0, which does not match the first check in the List 6, a_0 is not applicable to state s_0 and a_0 is not allowed to add to the search tree.

Table 16: A movie activity a_0 - pre and post context

variable name	data type	ontology type	constraints
pre-context			
l	string	start location	$l =$"350 Rue Emery, Montreal, QC, Canada"
t	date	start time point	$t = 19:35$
d	real	duration	$d = 88$
m	real	possessed money	$m = 100$
mv	boolean	movie	$mv = false$
post-context			
l	string	destination	$l =$"350 Rue Emery, Montreal, QC, Canada"
t	string	end time point	$t = 88 + 19:35$
d	real	duration	$d = 88$
m	real	possessed money	$m = 100 - 13$
mv	boolean	movie	$mv = true$

8. We continue to traverse the activity list. Suppose activity a_1 is a driving activity whose context description is shown in Table 17. According to the pre-context, a_1 is applicable to s_0.

   For the Driving Activity and Movie Activity more checking steps are required.

   - We have to constrain that the two driving activities cannot be executed one by one directly, which means we can only use this type of activity to connect other type of activities, but not suggest this type of activity to users.
   - And we program the planner to not let user to see one movie twice.

Table 17: A driving activity a_1 - pre and post context

variable name	data type	ontology type	constraints
pre-context			
l	string	start location	$l =$"H3K2S5"
t	string	end time point	$t = 19:00$
d	real	duration	$d = 10$
m	real	possessed money	$m = 100$
dr	boolean	driving	$dr = true \vee dr = false$
post-context			
l	string	destination	$l =$"350 Rue Emery, Montreal, QC, Canada"
t	string	end time point	$t = 19:00 + 10$
d	real	duration	$d = 10$
m	real	possessed money	$m = 100 - 10$
dr	boolean	driving	$dr = true$

9. After applying a_1 to s_0, a new state s_1 is added to the search tree, as shown in Figure 26, where the values of attributes are from Table 18, *i.e.* $s_1 = \{mv_1 = false, dn_1 = false, dr_1 = true, t_1 = 19:10, l_1 =$"350 Rue Emery, Montreal, QC, Canada"$, m_1 = 90\}$.

Table 18: Context for state s_1

variable name	data type	ontology type	value	constraints
l_1	string	start location	"350 Rue Emery, Montreal, QC, Canada"	$l_1 =$"350 Rue Emery, Montreal, QC, Canada"
m_1	real	possessed money	90	$m_1 = 90$
t_1	date	start time	19:10	$t_1 = 19:10$
mv_1	boolean	movie	false	$mv_1 = false$
dn_1	boolean	dining	false	$dn_1 = false$
dr_1	boolean	driving	true	$dr_1 = true$

10. And then check if s_1 is not visited, if so, record s_0 as the parent of s_1 and insert s_1 into *Qtemp*. If s_1 is visited, the whole branch rooted at s_1 is pruned from the tree.

11. We expend the next children of s_0 in the same way as we generate s_1 from s_0. Then, we can get one more state s_2 added to the search tree in Figure 27 and add the newly generated state into *Qtemp*. Similarly, we keep looking for all activities that are applicable to the initial state s_0 to generate new states and

add these states to the search tree as the children of s_0. After all applicable activities are applied to s_0, we get the search tree with level0 and level1 completed in Figure 28.

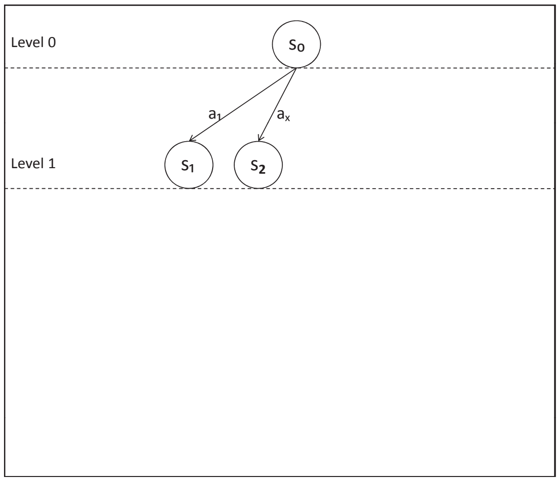


Figure 27: Search Tree Level 1.2

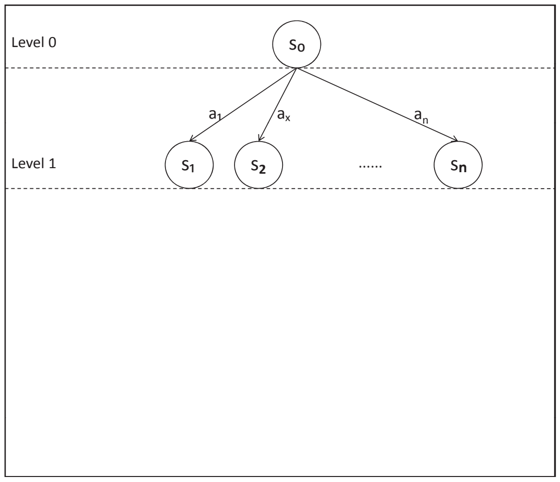


Figure 28: Search Tree Level 1.3

12. Now Q is empty, we sort the nodes in *Qtemp* according to the heuristic cost and keep only top k nodes. The higher the heuristic value is, the better the node is. For example, in our implementation, k is 10 and the heuristic cost of the top 10 nodes in the first level are listed as shown in Table 19.

Table 19: The Heuristic costs of the top 10 nodes in the first level

order	origin sequence	heuristic cost
0	7	0.8134
1	8	0.8115
2	9	0.7669
3	10	0.7487
4	16	0.6693
5	15	0.6680
6	14	0.6649
7	13	0.6635
8	12	0.6631
9	11	0.6624

13. Move all the elements from *Qtemp* to Q. Continue the next turn in while loop.

14. In the next turn, state s_1 becomes the current context. The program looks for all activities from the activity list that can be applicable to s_1. This time, activity a_0 (see Table 16 for the pre-context and post-context of a_0) is applicable to s_1 (see Table 18 for the context of s_1) because the location of s_1 is the same with the start location of a_0. After a_0 is applied to s_1, state s_{11} is added to the search tree, as shown in Figure 29, and the context of s_{11} is shown in Table 20. And so on, finally we'll get the search tree in Figure 30.

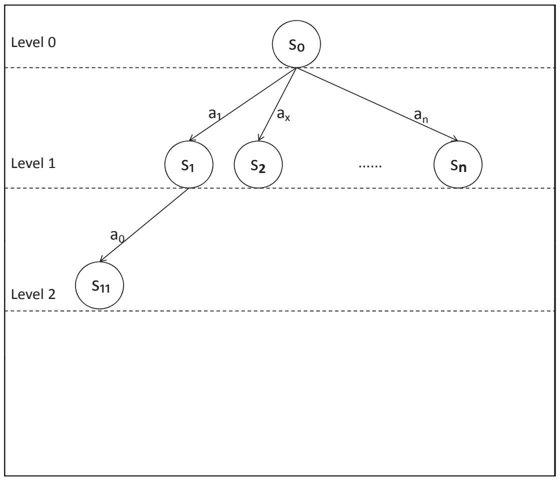


Figure 29: Search Tree Level 2.1

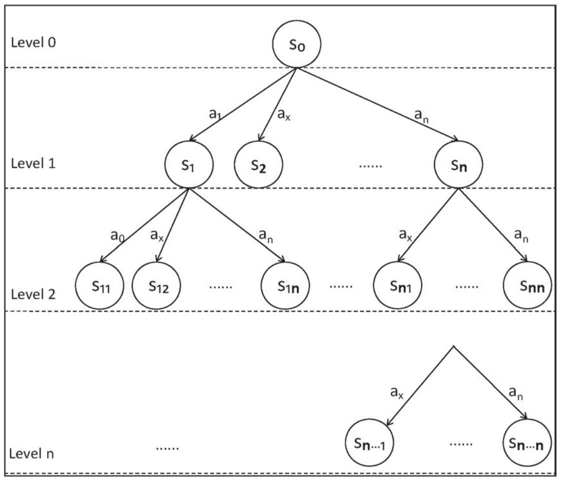


Figure 30: Search Tree Level N

Table 20: Context for state s_{11}

variable name	data type	ontology type	value	constraints
l_{11}	string	start location	"350 Rue Emery, Montreal, QC, Canada"	l_{11} ="350 Rue Emery, Montreal, QC, Canada"
m_{11}	real	possessed money	77	$m_{11} = 77$
t_{11}	date	start time	21:03	$t_{11} = 21:03$
mv_{11}	boolean	movie	true	$mv_{11} = true$
dn_{11}	boolean	dining	false	$dn_{11} = false$
dr_{11}	boolean	driving	true	$dr_{11} = true$

15. Every path in the search tree backward from leaf nodes to the root node is one of the solution. And all activities applied to the states in one path are what we suggest the user to do during the time.

6.7 System Optimization

Comparing to servers and desktop computers, the CPU power of smart phones is lower, the memory and the storage capacities of smart phones are smaller, and there are constraints on computing power and battery life on smart phones. However, the condition of mobile computing has much improved in recent years [Oh06], that's one of the key points we can run our application on a smart phone, unlike in common mobile applications, smart phones only act as client sides and are responsible for getting data from server sides and showing UI to users. In that kind of model, all real work is actually running on the server sides.

In order to run our application smoothly on a smart phone, we take the following steps to optimize the system performance in the ways as below:

- In the database layer, we choose JSON as the format to store data since JSON libraries have been embedded on Android and optimized to fit mobile applications, which allows to work fast with JSON files in the application layer of our system.

- In service mashup, we use jsoup [Hed12], a handy little library, to manipulate HTML files from Google showtime. When the Google map RESTful service is invoked, we require JSON as returned format rather than XML. Furthermore, we reasonably cut down the number of returned movie activities and restaurant activities.

- In service composition, we appropriately cut down the using of object-orientation technology in our algorithm, *e.g.* using *int* and / or *String* type in plenty of cyclical equal or unequal checking instead of properties of objects.

- We use cache mechanism in the system. In the UI layer, we store the previous user inputs and outputs in the cache. Once a user submits a request, if the current inputs match with the previous inputs, we return the previous outputs directly. In the database layer, we store all the activities generated from mashup phase. If the specified input, *i.e.* location, has been used as an input before, we compute the result list from the stored data such that mashup phase is skipped. Taking the advantage of the cache mechanism, we reduce repeating computing and improve the performance.

6.8 Testing Issues

We conduct tests on each individual module within the entertainment planner system as separate entities over HTC Incredible S and HTC Wildfire S. Once each individual module is tested thoroughly, the package is built together and tested as a whole. All known valid input is tested as well as known invalid input.

6.8.1 Unit Testing

The unit testing has been done with JUnit for Android. All results as shown in the following figures are tested on HTC Incredible S.

Service Mashup Testing

1. HTML Service Testing

   We conduct the test cases for the Google Show Time service as below. The result of test cases can be found in Figure 31, we can see that it takes about 15 seconds to get the values back, each test case invokes the service once.

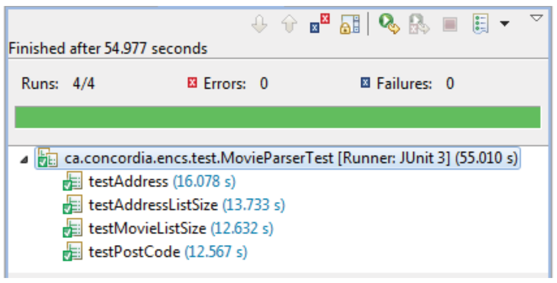
   (a) Invoking the service with a postcode "H3K1Z8". The test case executes successfully if there is no exception caught, otherwise the test case fails.

   (b) Invoking the service with an address "977 rue Sainte Catherine O., Montreal, QC, Canada". The test case executes successfully if there is no exception caught, otherwise the test case fails.

   (c) Invoking the service with an address identified by its postcode "H3K1Z8". The test case is successful if the size of *movieList* is greater than 0 after the service is invoked, since we can not specify how many movie activities are returned, otherwise the test case fails.

   (d) Invoking the service with an address identified by its postcode "H3K1Z8". The test case is successful if the size of *addressList* is 10 after the service is invoked, since we only request 10 movie theaters around, *i.e.* 10 addresses, otherwise the test case fails.

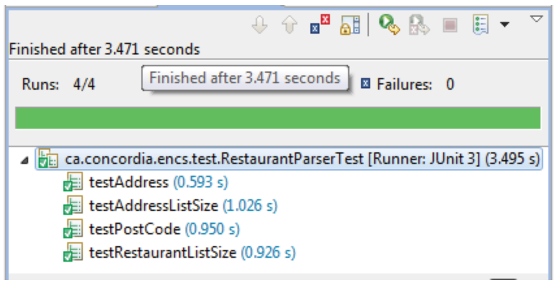
2. SOAP Service Testing


Finished after 54.977 seconds

Runs: 4/4 Errors: 0 Failures: 0

```
ca.concordia.encs.test.MovieParserTest [Runner: JUnit 3] (55.010 s)
    testAddress (16.078 s)
    testAddressListSize (13.733 s)
    testMovieListSize (12.632 s)
    testPostCode (12.567 s)
```

Figure 31: The result of HTML service testing


Finished after 3.471 seconds

Runs: 4/4 Failures: 0

```
ca.concordia.encs.test.RestaurantParserTest [Runner: JUnit 3] (3.495 s)
    testAddress (0.593 s)
    testAddressListSize (1.026 s)
    testPostCode (0.950 s)
    testRestaurantListSize (0.926 s)
```

Figure 32: The result of SOAP service testing

We conduct the test cases for the Bing SOAP search service as below. The result of test cases can be found in Figure 32, we can see that it takes about 1 second to get the values back, each test case invokes the service once.

(a) Invoking the service with a postcode "H3K1Z8". The test case executes successfully if there is no exception caught, otherwise the test case fails.

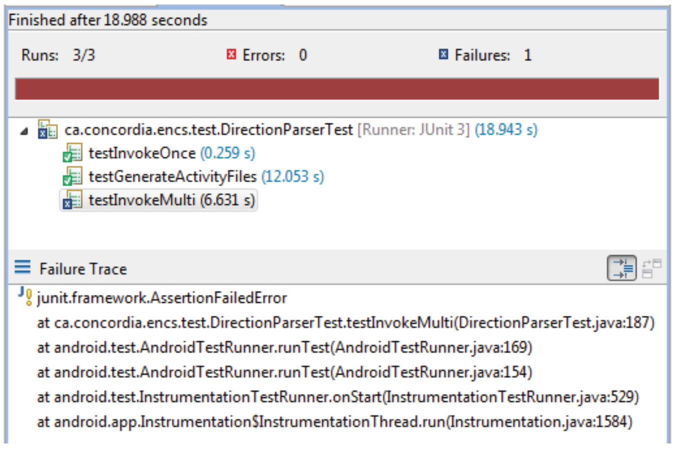
(b) Invoking the service with an address "977 rue Sainte Catherine O., Montreal, QC, Canada". The test case executes successfully if there is no exception caught, otherwise the test case fails.

(c) Invoking the service with an address identified by its postcode "H3K1Z8". The test case executes successfully if the size of *restaurantList* is 10 after the service is invoked, since we only request 10 restaurants around, otherwise the test case fails.

(d) Invoking the service with an address identified by its postcode "H3K1Z8". The test case executes successfully if the size of *addressList* is 10 after the service is invoked, since we only request 10 restaurants around, *i.e.* 10 addresses, otherwise the test case fails.

3. RESTful Service Testing

We conduct the test cases for the Google Maps RESTful service as below. The results can be found in Figure 33. The duration of running depends on how many invocations, since we have to wait between two invocations. For a specific invocation, the duration of running can be omitted.



```
Finished after 18.988 seconds
Runs: 3/3          Errors: 0          Failures: 1

ca.concordia.encs.test.DirectionParserTest [Runner: JUnit 3] (18.943 s)
    testInvokeOnce (0.259 s)
    testGenerateActivityFiles (12.053 s)
    testInvokeMulti (6.631 s)

Failure Trace
junit.framework.AssertionFailedError
at ca.concordia.encs.test.DirectionParserTest.testInvokeMulti(DirectionParserTest.java:187)
at android.test.AndroidTestRunner.runTest(AndroidTestRunner.java:169)
at android.test.AndroidTestRunner.runTest(AndroidTestRunner.java:154)
at android.test.InstrumentationTestRunner.onStart(InstrumentationTestRunner.java:529)
at android.app.Instrumentation$InstrumentationThread.run(Instrumentation.java:1584)
```

Figure 33: The result of RESTful service testing

(a) Invoking the service by passing two location2 $l1$ and $l2$. The test case executes successfully if the returned driving activity a is not $Null$ and $a.getLocation1().equals(l1)$ and $a.getLocation2().equals(l2)$, otherwise the test case fails.

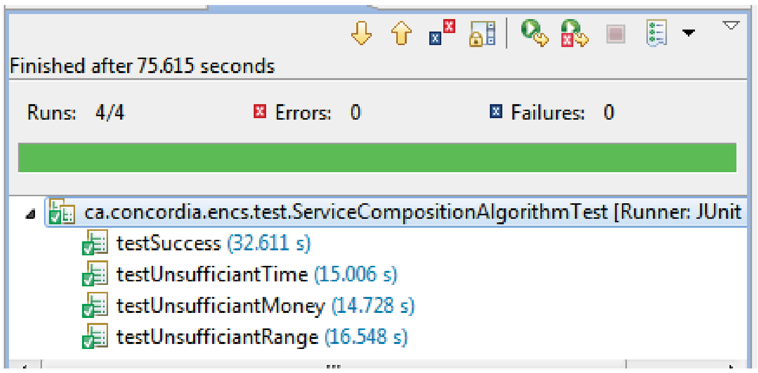
(b) Invoking the service by passing an address list containing 6 different addresses, 1200 milliseconds between two invocations. The test case executes successfully if the size of the returned driving activity list is 30 since the direction matters, *i.e.* from location $l1$ to location $l2$ is different from $l2$ to $l1$, otherwise the test case fails.

(c) Invoking the service by passing an address list containing 6 different addresses without any interval. The test case executes successfully if the size of the returned driving activity list is 30, otherwise the test case fails. This test case **fails** because of Google's limitation on invocation frequency.

Service Composition Testing

We conduct the test cases for the service composition as below. The result of test cases can be found in Figure 34, we can see that it takes about 30 seconds to have the successful result.

1. Invoking the service composition with the location identified by "H3K1Z8", location range 20 km, start time point 19:00, duration 180 minutes, and the money 50 dollars. The test should be successful if the size of result is 10, otherwise the test case fails.


Finished after 75.615 seconds

Runs: 4/4 Errors: 0 Failures: 0

```
ca.concordia.encs.test.ServiceCompositionAlgorithmTest [Runner: JUnit
    testSuccess (32.611 s)
    testUnsufficiantTime (15.006 s)
    testUnsufficiantMoney (14.728 s)
    testUnsufficiantRange (16.548 s)
```

Figure 34: The result of RESTful service testing

2. Invoking the service composition with the parameters as the test case above, except the money is not sufficient *i.e.* 10 dollars. The test should be successful if there is no plan returned, otherwise the test case fails.

3. Invoking the service composition with the parameters as the first test case, except the range is not large enough *i.e.* 1 km. The test should be successful if there is no plan returned, otherwise the test case fails.

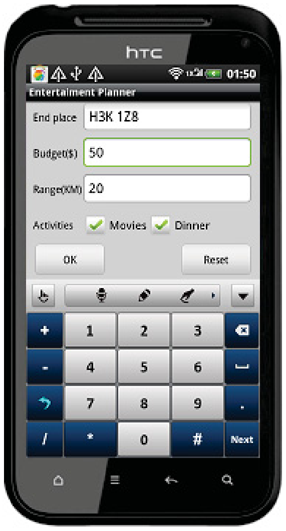
4. Invoking the service composition with the parameters as the test case above, except the duration is not sufficient *i.e.* 60 minutes. The test should be successful if there is no plan returned, otherwise the test case fails.

6.8.2 Compatibility Testing

We use two different models of Android phones, HTC Incredible S with the 4.0 inches screen on OS version 2.3.4 and HTC Wildfire S with 3.2 inches screen on OS version 2.3.3 (see Section 6.2.2), to test our application. All views of our application are scrollable, and the layout is designed in percentage. Comparing Figure 35 and Figure 36, we can see our application is compatible with different screen sizes even with the Keyboard shown. Moreover, it can be run on the different versions of OS.

6.8.3 Performance Testing

We use Traceview, a performance analysis tool comes with Android SDK, to analyze the performance of our application. Traceview depends on plenty of "debug data" recorded onto a smart phone's SD card to generate a graphic performance analysis


Entertainment Planner

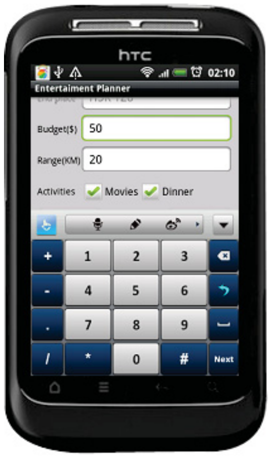
End place H3K 1Z8

Budget($) 50

Range(KM) 20

Activities ✓ Movies ✓ Dinner

Figure 35: HTC Incredible S


Entertainment Planner

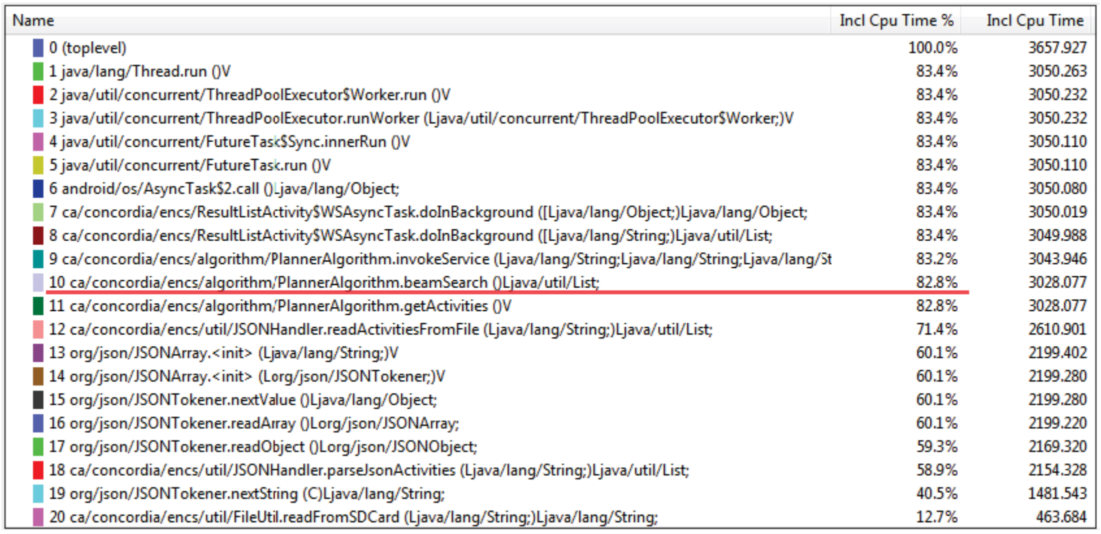
Budget($) 50

Range(KM) 20

Activities ✓ Movies ✓ Dinner

Figure 36: HTC Wildfire S

report, which largely slows down the application, so the CPU Time in the Figure is far from accuracy. However, we can consult the “CPU Time %” to acquire the percentage CPU time of each method in our application. As shown in Figure 37 *beamSearch* function uses 82.8% CPU Time of the application.



Name	Incl Cpu Time %	Incl Cpu Time
🟦 0 (toplevel)	100.0%	3657.927
🟩 1 java/lang/Thread.run ()V	83.4%	3050.263
🟥 2 java/util/concurrent/ThreadPoolExecutor$Worker.run ()V	83.4%	3050.232
🟦 3 java/util/concurrent/ThreadPoolExecutor.runWorker (Ljava/util/concurrent/ThreadPoolExecutor$Worker;)V	83.4%	3050.232
🟪 4 java/util/concurrent/FutureTask$Sync.innerRun ()V	83.4%	3050.110
🟨 5 java/util/concurrent/FutureTask.run ()V	83.4%	3050.110
🟦 6 android/os/AsyncTask$2.call ()Ljava/lang/Object;	83.4%	3050.080
🟩 7 ca/concordia/encs/ResultListActivity$WSAsyncTask.doInBackground ([Ljava/lang/Object;)Ljava/lang/Object;	83.4%	3050.019
🟥 8 ca/concordia/encs/ResultListActivity$WSAsyncTask.doInBackground ([Ljava/lang/String;)Ljava/util/List;	83.4%	3049.988
🟦 9 ca/concordia/encs/algorithm/PlannerAlgorithm.invokeService (Ljava/lang/String;Ljava/lang/String;Ljava/lang/St	83.2%	3043.946
🟪 10 ca/concordia/encs/algorithm/PlannerAlgorithm.beamSearch ()Ljava/util/List;	82.8%	3028.077
🟩 11 ca/concordia/encs/algorithm/PlannerAlgorithm.getActivities ()V	82.8%	3028.077
🟥 12 ca/concordia/encs/util/JSONHandler.readActivitiesFromFile (Ljava/lang/String;)Ljava/util/List;	71.4%	2610.901
🟪 13 org/json/JSONArray.<init> (Ljava/lang/String;)V	60.1%	2199.402
🟫 14 org/json/JSONArray.<init> (Lorg/json/JSONTokener;)V	60.1%	2199.280
⬛ 15 org/json/JSONTokener.nextValue ()Ljava/lang/Object;	60.1%	2199.280
🟦 16 org/json/JSONTokener.readArray ()Lorg/json/JSONArray;	60.1%	2199.220
🟩 17 org/json/JSONTokener.readObject ()Lorg/json/JSONObject;	59.3%	2169.320
🟥 18 ca/concordia/encs/util/JSONHandler.parseJsonActivities (Ljava/lang/String;)Ljava/util/List;	58.9%	2154.328
🟦 19 org/json/JSONTokener.nextString (C)Ljava/lang/String;	40.5%	1481.543
🟪 20 ca/concordia/encs/util/FileUtil.readFromSDCard (Ljava/lang/String;)Ljava/lang/String;	12.7%	463.684

Figure 37: Traceview Time Line

Supposing we have 10 movie theaters, 10 restaurants, the number of the driving activities between them is $2 \times 10 \times 10$, since the driving activity dr_1 (from a_1 to a_2) is different from dr_2 (from a_2 to a_1). The average execution time for each module of the application are listed in the Table 21. Please note, the times shown in the figures in Section 6.8.1 are little bit longer than in the real devices.

Table 21: Test Results

Module	Average Execution Time (second) on Wildfire S	Average Execution Time (second) on Incredible S
Mashup	$1.2 \times 2 \times 10 \times 10 + 25$	$1.2 \times 2 \times 10 \times 10 + 14$
Composition	65–80	22–26
UI	< 3	< 2

6.9 Appendices

6.9.1 Packaging and installation issues

The Software is packaged and distributed as an applications installation package. It is to be installed to the mobile devices running the Android Operating System version 2.3.3 and above.

6.9.2 Legal Considerations

We will be using the Android Software Development Kit (SDK) in accordance to the Android SDK License Agreement distributed by Google (Copyright holder of the Android SDK). This agreement grants us as developers “limited, worldwide, royalty-free, non-assignable and non-exclusive license to use the SDK solely to develop applications to run on the Android platform” [And12].

Chapter 7

Conclusion

This thesis begins with problem and motivation. Nowadays, smart phones become the most common type of mobile computing device. The ubiquity of sensing and computing power make smart phones have the advantages over normal cell phones. Web service discovery and composition are studied intensively nowadays. In this thesis, we want to investigate the possibilities to do Web service discovery and composition over smart phones. The Web service discovery and composition can be highly context based that it gives us different solutions based the the location, time, and user profile. Therefore, it has a lot advantages, if its computation power allows us to do that kind of complicated tasks.

Different from any desktop applications, the smart phone applications could be highly adaptive to contexts, *e.g.* location, identity, and time. In order to model the context information, we propose a constraint model for context representation (Chapter 3). We use constraint sets to represent the context of the mobile application, where a constraint gives a value or a range to entities in mobile applications that can either be automatically detected by the mobile system or provided by the user. Our constraint-based context model having the power to express the context for the mobile application paves the way for the mashup procedure.

Smart phones can discover different types of services over the Internet. In order to be able to compose different types of services, we use the mashup mechanism to uniform the format of the data coming from different types of data services (Section 4.6). We are able to mashup services using three different protocols, *i.e.* SOAP services, RESTful services, and HTML services.

Based on the context models and service mashup, we propose an AI-planning based *service composition* approach to develop an entertainment planner (Chapter 5) since service composition can be modeled as an AI planning problem [YZ08, ZY08]. Our work is a feasibility study for AI techniques on reduced environment. We use a beam search algorithm to solve service composition problem (Section 5.2). As far as we know, beam search is a first implementation of service composition on the smart phone environment. Beam search uses breadth-first search to build its search tree. At each level of the tree, it only stores a predetermined number of states (called beam width) in increasing order of heuristic cost. Therefore, the beam search algorithm optimizes the choice of states when constructing a search tree. Also, the beam width bounds the memory required to perform the search which suits the computation power of mobile phones easily.

Taking advantage of service composition approach, we design an entertainment planner for smart phones with the Google Android operating system (Chapter 6). The entertainment planner implements context modeling, service discovery mashup and service composition over smart phones. In order to optimize the system performance, the entertainment planner is capable of storing the results for the beam search. This is an exploratory work illustrating that the capacity of a smart phone is sufficient for mashup calculations, even for the three different protocols, and that it is possible to store the results on the limited phone storage without disabling regular functionality. The system follows the three-tier architectural style and is organized into three layers: the interface layer, the application layer and the storage layer(Section 6.3). The three-tier architecture style is used because it not only separates the user interface and the meta data, but also provides an application logic layer. The application layer provides a middle layer that allows the data files and the UI components to be loosely coupled, and that also makes the system more maintainable and reusable. It is possible to port the architecture to other tasks easily.

The current limitation to our approach is the chain of mashuped services is generated manually as it is not possible to fully automatically interpret the data format of RESTful services and HTML services.

Appendix A

Planner Beam Search Algorithm

```
package ca.concordia.encs.algorithm;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

import android.content.Context;
import ca.concordia.encs.models.Activity;
import ca.concordia.encs.models.Address;
import ca.concordia.encs.models.Constraint;
import ca.concordia.encs.models.Global;
import ca.concordia.encs.models.Proposition;
import ca.concordia.encs.models.SearchTreeNode;
import ca.concordia.encs.models.Solution;
import ca.concordia.encs.models.State;
import ca.concordia.encs.services.DirectionParser;
import ca.concordia.encs.services.MovieParser;
import ca.concordia.encs.services.RestaurantParser;
import ca.concordia.encs.util.AlgorithmUtil;
import ca.concordia.encs.util.JSONHandler;
import ca.concordia.encs.util.NonRepeatList;

/**
 * @author Jia Ning Wang(Apple)
 */
public class PlannerAlgorithm {

  private List<Activity> movieList =
      new ArrayList<Activity>();
  private List<Activity> restaurantList =
      new ArrayList<Activity>();
  private List<Activity> directionList =
      new ArrayList<Activity>();
  private List<Activity> activities =
      new ArrayList<Activity>();

  private List<Address> addressList =
      new NonRepeatList<Address>();

  private JSONHandler jsonHandler;

  private int beamWidth = 10;
  private State s0 = new State();
  private List<Proposition> goalProposition =
      new ArrayList<Proposition>();
  private List<Constraint> constraints =
      new ArrayList<Constraint>();

  public PlannerAlgorithm(Context context) {
    jsonHandler = new JSONHandler(context);
  }

  /**
   * The method is invoked by ResultListActivity
   *  to get the plans
   */
  public List<Solution> invokeService(String add,
      String money, String distance, String maxTime,
      String time, boolean mv, boolean dn) {

    double maxMoney = Double.parseDouble(money);
    double maxDistance = Double.parseDouble(distance);
    double dmaxTime = Double.parseDouble(maxTime) ;
    setS0(add, maxMoney, time);
    setConstraints(maxMoney, maxDistance, dmaxTime);
    setGoalProposition(mv,dn);

    return this.beamSearch();
  }

  /**
   * set goal proposition
   */
  private void setGoalProposition(boolean mv, boolean dn) {

    Proposition p1 = new Proposition("Movie", mv);
    Proposition p2 = new Proposition("Dining", dn);

    goalProposition.add(p1);
    goalProposition.add(p2);
  }

  /**
   * set constraints according to the input
   */
  private void setConstraints(double maxMoney,
      double maxDistance, double maxTime) {
    if (maxTime > 0) {
      Constraint conDuration = new Constraint("Time",
          maxTime, 0);
      constraints.add(conDuration);
    }
    if (maxDistance > 0) {
      Constraint conDistance = new Constraint(
          "Location", maxDistance * 1000, 0);
      constraints.add(conDistance);
    }
    if (maxMoney > 0) {
      Constraint conCost = new Constraint("Cost",
          maxMoney, 0);
      constraints.add(conCost);
    }
  }
```

```
/**
 * set initial state
 */
public void setS0(String add, double maxMoney,
    String startTime) {
  Address address = new Address();
  address.setPostCode(add);
  s0.setLocation(address);

  s0.setCurrentTime(AlgorithmUtil.getDate(startTime));
  s0.setMoney(maxMoney);

  Proposition p01 = new Proposition("Movie", false);
  Proposition p02 = new Proposition("Dining", false);

  s0.getPropositions().add(p01);
  s0.getPropositions().add(p02);
}

/**
 * beam search algorithm
 */
private List<Solution> beamSearch() {

  this.getActivities();

  List<Solution> results = new ArrayList<Solution>();

  List<SearchTreeNode> searchTree =
      new ArrayList<SearchTreeNode>();
  SearchTreeNode root = new SearchTreeNode();
  root.setSequence(0);
  root.setState(s0);

  searchTree.add(root);

  Queue<SearchTreeNode> beamQueue =
      new LinkedList<SearchTreeNode>();
  Queue<SearchTreeNode> leveledQueue =
      new LinkedList<SearchTreeNode>();

  root.setVisited(true);
  beamQueue.offer(root);

  while (!beamQueue.isEmpty()) {
    SearchTreeNode currentNode = beamQueue.poll();

    if (currentNode == null) {
      continue;
    }

    if (this.propositionsIncluded(goalProposition,
        currentNode.getState())) {
      Solution solution = this
          .retrieveSolution(currentNode);
      results.add(solution);
      System.out.println("Results␣number:"
          + results.size());

      if (results.size() >= beamWidth) {
        break;
      }
    }

    for (Activity a : this.activities) {
      if (this.applicable(currentNode.getState(),
          a)
          && this.specialActivityApplicable(
              currentNode, a)) {
        State s1 = this.transit(
            currentNode.getState(), a);
        if (this.satisfyConstraint(s1)) {

          SearchTreeNode tmpNode = new SearchTreeNode();
          tmpNode.setActivity(a);
          tmpNode.setParent(currentNode);
          tmpNode.setState(s1);
          tmpNode.setSequence(searchTree
              .size());
          SearchTreeNode visitedNode = this
              .getVisitedNode(searchTree,
                  tmpNode);
          if (visitedNode == null) {
            tmpNode.setVisited(true);

            searchTree.add(tmpNode);
            leveledQueue.offer(tmpNode);
          } else {
            this.pruneBranch(searchTree,
                visitedNode);
          }
        }
      }
    }

    System.out.println("before␣beamQueue="
        + beamQueue.size() + ",␣leveledQueue="
        + leveledQueue.size());

    this.resetQueues(beamQueue, leveledQueue);

    System.out.println("after␣queue.size="
        + beamQueue.size() + ",␣leveledQueue="
        + leveledQueue.size() + "\n");

  }

  return results;
}

/**
 * get the needed activities of the algorithm
 */
private void getActivities() {

  this.addressList = jsonHandler
      .readAddressesFromFile();
```

```
/*
 * check if the location in the database already
 */
if (addressList.contains(s0.getLocation())) {

  /*
   * get movie activities from json formatted database
   */
  this.movieList = jsonHandler
      .readActivitiesFromFile(Global.MOVIE_FILE_NAME);

  /*
   * get dining activities from json formatted database
   */
  this.restaurantList = jsonHandler
      .readActivitiesFromFile(Global.RESTAURANT_FILE_NAME);

  /*
   * get driving activities from json formatted database
   */
  this.directionList = jsonHandler
      .readActivitiesFromFile(Global.DIRECTION_FILE_NAME);

  if (movieList.size() != 0
      && restaurantList.size() != 0
      && directionList.size() != 0) {
    activities.addAll(movieList);
    activities.addAll(restaurantList);
    activities.addAll(directionList);

    return;
  }
}

getMovieActivitiesFromWS();
getRestaurantActivitiesFromWS();
getDirectionActivitiesFromWS();

activities.addAll(movieList);
activities.addAll(restaurantList);
```

```
  activities.addAll(directionList);

  List<Address> jsonList = new NonRepeatList<Address>(
      addressList);
  for (Address a : addressList) {
    if (jsonList.contains(a)) {
      jsonList.remove(a);
    }
  }
  jsonHandler.createOrUpdateAddressFile(jsonList);
}

/**
 * invoke the Google Show Time HTML service to
 * get movie activities
 */
private void getMovieActivitiesFromWS() {

  MovieParser mParser = new MovieParser(
      s0.getLocation(), addressList);
  try {
    movieList = mParser.generateActivityFiles();
  } catch (Exception e) {
    e.printStackTrace();
  }

}

/**
 * invoke the Bing Map SOAP service to
 * get dining activities
 */
private void getRestaurantActivitiesFromWS() {
  RestaurantParser rParser = new RestaurantParser(
      s0.getLocation(), addressList);
  try {
    restaurantList = rParser
        .generateActivityFiles();
  } catch (Exception e) {
    e.printStackTrace();
  }
}

/**
 * invoke the Google Maps RESTful service to
 * get driving activities
 */
private void getDirectionActivitiesFromWS() {

  addressList.add(s0.getLocation());

  Address downtown = new Address();
  downtown.setPostCode(Global.DOWNTOWN_POSTCODE);
  if (!addressList.contains(downtown)) {
    addressList.add(downtown);
  }

  DirectionParser dParser = new DirectionParser(
      addressList);
  try {
    directionList = dParser.generateActivityFiles();
  } catch (Exception e) {
    e.printStackTrace();
  }
}

/**
 * get a solution from the search tree node
 * backward to the root
 */
private Solution retrieveSolution(SearchTreeNode node) {
  Solution solution = new Solution();

  SearchTreeNode n = new SearchTreeNode();
  n.setActivity(node.getActivity());
  n.setParent(node.getParent());
  n.setSequence(node.getSequence());
  n.setState(node.getState());
  n.setVisited(node.isVisited());
  for (Activity a : this.activities) {
    if (a.getType() == 3 // Direction
        && a.getLocation1().equals(
            node.getState().getLocation())
        && a.getLocation2().equals(
            s0.getLocation())) {
      solution.add(a);
      break;
    }
  }

  while (n.getParent() != null) {
    solution.add(n.getActivity());

    SearchTreeNode nod = new SearchTreeNode();
    nod.setActivity(n.getParent().getActivity());
    nod.setParent(n.getParent().getParent());
    nod.setSequence(n.getParent().getSequence());
    nod.setState(n.getParent().getState());
    nod.setVisited(n.getParent().isVisited());

    n = nod;
  }

  return solution;
}

/**
 * apply an activity to a state to get new state
 */
private State transit(State s, Activity a) {
  State s1 = new State();
  s1.setMoney(s.getMoney() - a.getCost());
  s1.setLocation(a.getLocation2());

  if (a.getType() == 2) {
    s1.setCurrentTime(AlgorithmUtil.addMinutes(
        a.getStartTime(), s.getCurrentTime(),
        a.getDuration()));
  } else {
    s1.setCurrentTime(AlgorithmUtil.addMinutes(
        s.getCurrentTime(), a.getDuration()));
  }

  s1.setPropositions(getState1Propositions(s, a));
  s1.setDistance(getDistanceBetween(s0.getLocation(),
      s1.getLocation()));

  return s1;
}

/**
 * get the visited search tree node
 */
private SearchTreeNode getVisitedNode(
    List<SearchTreeNode> searchTree,
    SearchTreeNode node) {
  for (SearchTreeNode s : searchTree) {
    if (s.getParent() == null
        || node.getParent() == null) {
      if (s.getParent() == null
          && node.getParent() == null) {
        if (s.getActivity().equals(
            node.getActivity())) {
          return node;
        }
      }
    } else {
      if (s.getParent().getSequence() == node
          .getParent().getSequence()
          && s.getActivity().equals(
              node.getActivity())) {
        return node;
      }
    }
  }
  return null;
}

/**
```

```
 * prune a branch from the search tree
 */
private void pruneBranch(
    List<SearchTreeNode> searchTree,
    SearchTreeNode node) {
  for (SearchTreeNode s : searchTree) {
    if (s.getSequence() == node.getSequence()) {
      deleteSon(searchTree, s);
      break;
    }
  }
}

/**
 * delete the son of the specified node of the search tree
 */
private void deleteSon(List<SearchTreeNode> searchTree,
    SearchTreeNode node) {
  for (SearchTreeNode s : searchTree) {
    if (s.getParent() != null) {
      if (s.getParent().getSequence() == node
          .getSequence()) {
        deleteSon(searchTree, s);
      }
    }
  }

  searchTree.remove(node);
}

/**
 * check if a state satisfy the constraints
 */
private boolean satisfyConstraint(State s) {
  int i = 0;
  for (Constraint c : this.constraints) {
    if (c.getName().equals("Cost")) {
      if (c.satisfy(c.getName(), s.getMoney())) {
        i++;
      }
    } else if (c.getName().equals("Location")) {
      int distance = this.getDistanceBetween(
          s.getLocation(), s0.getLocation());
      if (distance != -1) {
        if (c.satisfy(c.getName(), distance)) {
          i++;
        }
      }
    } else if (c.getName().equals("Time")) {
      if (c.satisfy(
          c.getName(),
          AlgorithmUtil.getMinuteDiff(
              s.getCurrentTime(),
              s0.getCurrentTime()))) {
        i++;
      }
    }
  }

  if (i == constraints.size()) {
    return true;
  }
  return false;
}

/**
 * check if an activity is applicable to a state
 */
private boolean applicable(State s, Activity a) {
  if (!s.getLocation().equals(a.getLocation1())) {
    return false;
  }

  if (propositionsIncluded(a.getPreConditions(), s)) {
    return true;
  }

  return false;
}
```

```
private boolean specialActivityApplicable(
    SearchTreeNode currentNode, Activity a) {
  if (currentNode.getActivity() == null) {
    return true;
  }

  if (a.getType() == 3) {// Direction
    if (currentNode.getActivity().getType() == 3) {
      return false;
    } else {
      return true;
    }
  }

  if (a.getType() == 2) {
    if (AlgorithmUtil
        .timeBefore(a.getStartTime(),
            currentNode.getState()
                .getCurrentTime())) {
      return false;
    }
    Solution solution = retrieveSolution(currentNode);
    for (int i = 0; i < solution.size(); i++) {
      if (solution.get(i).getType() == 2) {
        Activity mv = solution.get(i);
        if (mv.getMovieName().equals(
            a.getMovieName())) {
          return false;
        }
      }
    }
  }

  return true;
}

/**
 * reset the queue according to the beam width
 */
private void resetQueues(
    Queue<SearchTreeNode> beamQueue,
    Queue<SearchTreeNode> leveledQueue) {
  if (beamQueue.isEmpty()) {
    while (!leveledQueue.isEmpty()) {
      beamQueue.offer(leveledQueue.poll());
    }
    setQueueAsBeamWidth(beamQueue);
  }
}

private void setQueueAsBeamWidth(
    Queue<SearchTreeNode> queue) {
  ArrayList<SearchTreeNode> list =
      new ArrayList<SearchTreeNode>();

  while (!queue.isEmpty()) {
    SearchTreeNode n = queue.poll();
    int i = 0;

    for (Activity a : this.activities) {
      if (applicable(n.getState(), a)
          && specialActivityApplicable(n, a)) {
        State s1 = transit(n.getState(), a);
        if (satisfyConstraint(s1)) {
          i++;
        }
      }
    }

    n.getState().setNextActivities(i);

    list.add(n);
  }

  sortOnHuristic(list);

  int i = 0;
  for (SearchTreeNode node : list) {
    if (++i > beamWidth) {
      break;
    }
    queue.offer(node);
  }
}

/**
 * sort the search tree according to heuristic cost
 */
private void sortOnHuristic(
    ArrayList<SearchTreeNode> list) {
  if (list == null || list.size() == 0) {
    return;
  }

  State state0 = list.get(0).getState();
  double mmin = state0.getMoney();
  double mmax = state0.getMoney();
  double dmin = state0.getDistance();
  double dmax = state0.getDistance();
  double nmin = state0.getNextActivities();
  double nmax = state0.getNextActivities();

  double md, dd, nd;

  for (int i = 1; i < list.size(); i++) {
    State state = list.get(i).getState();
    if (state.getMoney() > mmax) {
      mmax = state.getMoney();
    }
    if (state.getMoney() < mmin) {
      mmin = state.getMoney();
    }
    if (state.getDistance() > dmax) {
      dmax = state.getDistance();
    }
    if (state.getDistance() < dmin) {
      dmin = state.getDistance();
    }
    if (state.getNextActivities() > nmax) {
      nmax = state.getNextActivities();
    }
    if (state.getNextActivities() < nmin) {
      nmin = state.getNextActivities();
    }
  }

  for (int i = 0; i < list.size(); i++) {
    State state = list.get(i).getState();

    if (mmax != mmin) {
      md = (state.getMoney() - mmin)
          / (mmax - mmin);
    } else {
      md = 1;
    }

    if (dmax != dmin) {
      dd = (dmax - state.getDistance())
          / (dmax - dmin);
    } else {
      dd = 1;
    }

    if (nmax != nmin) {
      nd = (state.getNextActivities() - nmin)
          / (nmax - nmin);
    } else {
      nd = 1;
    }

    list.get(i).setHeuristic(
        md * 0.6 + dd * 0.2 + nd * 0.2);
  }

  Collections.sort(list,
      new Comparator<SearchTreeNode>() {
        public int compare(SearchTreeNode i,
            SearchTreeNode j) {
          return (int) ((j.getHeuristic() - i
              .getHeuristic()) * 10000);
        }
      });
}

/**
 * check if the state's propostions are included
 * in goal propositions
 */
private boolean propositionsIncluded(
    List<Proposition> goalpros, State s) {
  List<Proposition> statepros = s.getPropositions();

  int i = 0;
  for (Proposition goal : goalpros) {
    if (statepros.contains(goal)) {
      i++;
    }
  }

  if (i == goalpros.size()) {
    return true;
  }
  return false;
}

private List<Proposition> getState1Propositions(
    State s, Activity a) {
  List<Proposition> newPros = new ArrayList<Proposition>();

  newPros.addAll(s.getPropositions());

  for (Proposition add : a.getPositiveEffects()) {
    if (!newPros.contains(add)) {
      newPros.add(add);
    }
  }

  for (Proposition sub : a.getNegativeEffects()) {
    if (newPros.contains(sub)) {
      newPros.remove(sub);
    }
  }

  return newPros;
}

private int getDistanceBetween(Address add1,
    Address add2) {
  for (Activity activity : this.directionList) {
    Activity direction = activity;
    if (direction.getLocation1().equals(add1)
        && direction.getLocation2()
            .equals(add2)) {
      return direction.getDistance();
    }
  }
  return -1;
}
}
```

Bibliography

[AGS+93] Norman Adams, Rich Gold, Bill N. Schilit, Michael Tso, and Roy Want. An infrared network for mobile computers. In *In Proceedings USENIX Symposium on Mobile & Location-independent Computing*, pages 41–52, 1993.

[ama12] amazon.com. Amazon elastic compute cloud (amazon ec2). `http://aws.amazon.com/ec2/`. Retrieved June 19, 2012, 2012.

[And12] Android.com. Google android developer. `http://developer.android.com/develop/index.html`. Retrieved June 19, 2012, 2012.

[BBC97] Peter J. Brown, John D. Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, October 1997.

[BBH+10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modeling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, 2010.

[BKH11] Sergey Balandin, Yevgeni Koucheryavy, and Honglin Hu, editors. *Smart spaces and next generation wired/wireless networking 11th international conference 11th International Conference, NEW2AN 2011, and 4th Conference on Smart Spaces, ruSMART 2011 St. Petersburg, Russia, August 22-25, 2011: Proceedings*, Heidelberg, 2011. Springer.

[CFJ03] Harry Chen, Tim Finin, and Anupam Joshi. Using OWL in a pervasive computing broker. In *Proceedings of Workshop on Ontologies in Open Agent Systems (AAMAS 2003)*, 2003.

[CK00] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth, November 2000.

[CMD99] Keith Cheverst, Keith Mitchell, and Nigel Davies. Design of an object model for a context sensitive tourist guide. *Computers and Graphics*, 23(6):883–891, 1999.

[CW10] Peter J. Curwen and Jason Whalley. *Mobile telecommunications in a high speed world industry structure, strategic behaviour and socio-economic impact*. Gower Publishing Ltd, Farnham, 2010.

[Dan10] Guo Dan. New ideas for web service discovery-ontology-based prototype system of service search engine. In *2nd International Conference on Software Technology and Engineering*, volume 2, pages 407–409, 2010.

[Del12] Delicous.com. Delicious APIs. `http://www.delicious.com/developers` Retrieved June 19, 2012, 2012.

[Dey01] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing, Special issue on Situated Interaction and Ubiquitous Computing*, 5(1), 2001.

[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*, VLDB '04, pages 372–383. VLDB Endowment, 2004.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, 2000.

[Gol11] Paul Golding. *Connected services: a guide to the Internet technologies shaping the future of mobile services and operators*. John Wiley & Sons, Chichester, West Sussex, 2011.

[Goo12a] Google. Google. `http://www.google.com`. Retrieved June 19, 2012, 2012.

[Goo12b] Google. Google maps API family. `https://developers.google.com/maps/`. Retrieved June 19, 2012, 2012.

[Hal01] Terry Halpin. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufman Publishers, San Francisco, 2001.

[Hed12] Jonathan Hedley. jsoup. `http://jsoup.org/`. Retrieved June 19, 2012, 2012.

[HIR02] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In *Proceedings of the First International Conference on Pervasive Computing*, Pervasive '02, pages 167–180, London, UK, UK, 2002. Springer-Verlag.

[HIR03] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Generating context management infrastructure from high-level context models. In *Industrial Track Proceedings of the 4th International Conference on Mobile Data Management (MDM2003)*, pages 1–6, Melbourne/Australia, January 2003.

[IBM08] IBM. Restful web services. `http://www.ibm.com/developerworks/webservices/library/ws-restful/`. Retrieved June 19, 2012, 2008.

[IRRH03] Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, and Karen Henricksen. Experiences in using cc/pp in context-aware systems. In *In Proc. of the Intl. Conf. on Mobile Data Management (MDM*, pages 247–261. Springer, 2003.

[iSUG12] iPhone 4S User Guide. idownloadblog. `http://www.apple.com/iphone/features/siri.html`. Retrieved June 19, 2012, 2012.

[MB97] John McCarthy and Saša Buvač. Formalizing context(expanded notes). In *Working Papers of the AAAI Fall Symposium on Context in Knowledge Representation and Natural Language*, pages 99–135, Menlo Park, California, December 1997.

[McC93] John McCarthy. Notes on formalizing context. In *Proceedings of the 13th international joint conference on Artifical intelligence - Volume 1*, IJCAI'93, pages 555–560, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[Mic12a] Microsoft. Bing. `http://www.bing.com`. Retrieved June 19, 2012, 2012.

[Mic12b] Microsoft. Bing services 2.0. `http://msdn.microsoft.com/en-us/library/dd877956.aspx`. Retrieved June 19, 2012, 2012.

[Mon12] Tourisme Montréal. Events. `http://www.tourisme-montreal.org/What-To-Do/Events`. Retrieved June 19, 2012, 2012.

[mos11] mosabua@gmail.com. ksoap2-android. `http://code.google.com/p/ksoap2-android/`. Retrieved June 19, 2012, 2011.

[MSXZ10] Chao Ma, Meina Song, Ke Xu, and Xiaoqi Zhang. Web service discovery research and implementation based on semantic search engine. In *2nd Symposium on Web Society*, pages 672–677, Beijing, China, 2010.

[ODW11] Mohammad S. Obaidat, Mieso Denko, and Isaac Woungang, editors. *Pervasive computing and networking*. John Wiley & Sons, Hoboken, NJ, 2011.

[Oh06] Sangyoon Oh. *Web Service Architecture For Mobile Computing*. Phd dissertation, University of Indiana, 2006.

[Ora06] Oracle. Restful web services. `http://www.oracle.com/technetwork/articles/javase/index-137171.html`. Retrieved June 19, 2012, 2006.

[Pas98] Jason Pascoe. Adding generic contextual capabilities to wearable computers. In *2nd International Symposium on Wearable Computers (ISWC 1998)*, pages 92–99, 1998.

[Per10] Technically Personal. Use permissions to secure your private data from android apps. `http://techpp.com/2010/07/30/android-apps-permissions-secure-private-data/`. Retrieved June 19, 2012, 2010.

[Pie11] Samuel Pierre. *Next Generation Mobile Networks and Ubiquitous Computing*. Information Science Publishing, Hershey, 2011.

[Rya99] Nick Ryan. ConteXtML: Exchanging contextual information between a mobile client and the fieldnote server. 1999.

[SAG$^+$93] Bill N. Schilit, Norman Adams, Rich Gold, Michael M. Tso, and Roy Want. The ParcTab mobile computing system, 1993.

[SAW94] Bill N. Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, US, 1994.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. *Computers and Graphics*, 23(6):893–901, 1999.

[Sch95] William Noah Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.

[See12] Seekda! Web services search engine. `http://webservices.seekda.com`. Retrieved June 19, 2012, 2012.

[SLP04] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing*, Nottingham/England, 2004.

[SLPF03] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Cool: A context ontology language to enable contextual interoperability. In J.-B. Stefani, I. Demeure, and D. Hagimont, editors, *LNCS 2893: Proceedings of 4th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS2003)*, volume 2893 of *Lecture Notes in Computer Science (LNCS)*, pages 236–247, Paris/France, November 2003. Springer Verlag.

[SMLP01] Michael Samulowitz, Florian Michahelles, and Claudia Linnhoff-Popien. CAPEUS: An architecture for context-aware selection and execution of services. In *Proceedings of the IFIP TC6 / WG6.1 Third International Working Conference on New Developments in Distributed Applications and Interoperable Systems*, pages 23–40, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

[Sob10] Tarek Sobh, editor. *Innovations and advances in computer sciences and engineering*, Dordrecht, 2010. Springer.

[Str03] Thomas Strang. *Service Interoperability in Ubiquitous Computing Environments*. PhD thesis, Ludwig-Maximilians-UniversityMunich, October 2003.

[TAY10] Asoke K. Talukder, Hasan Ahmed, and Roopa R. Yavagal. *Mobile computing: technology, applications, and service creation (2nd ed.)*. Tata McGraw Hill, New Delhi, 2010.

[W3C04a] W3C. Owl-s: Semantic markup for web services. `http://www.w3.org/Submission/OWL-S/`. Retrieved June 19, 2012, 2004.

[W3C04b] W3C. Web services architecture. `http://www.w3.org/TR/ws-arch/`. Retrieved June 19, 2012, 2004.

[W3C07a] W3C. Semantic annotations for wsdl and xml schema (sawsdl). `http://www.w3.org/TR/sawsdl/`. Retrieved June 19, 2012, 2007.

[W3C07b] W3C. Web services description language (wsdl) version 2.0. `http://www.w3.org/TR/wsdl20/`. Retrieved June 19, 2012, 2007.

[Wik12a] Wikipedia. Android (operating system). `http://en.wikipedia.org/wiki/Android_(operating_system)`. Retrieved June 19, 2012, 2012.

[Wik12b] Wikipedia. Wdsl. `http://en.wikipedia.org/wiki/Web_Services_Description_Language`. Retrieved June 19, 2012, 2012.

[Wik12c] Wikipedia. Web service. `http://en.wikipedia.org/wiki/Web_service`. Retrieved June 19, 2012, 2012.

[WZGP04] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using OWL. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, PERCOMW '04, pages 18–22, Washington, DC, USA, 2004.

[Yah12] Yahoo. Yahoo! `http://search.yahoo.com`. Retrieved June 19, 2012, 2012.

[YZ08] Yuhong Yan and Xianrong Zheng. A planning graph based algorithm for semantic web service composition. In *CEC/EEE*, pages 339–342, Washington, DC, 2008.

[zA97] 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, Rio de Janeiro, February 1997.

[Zel10] Marvin Zelkowitz, editor. *Advances in Computers, Volume 78: Improving the Web*. Academic Press, Amsterdam, 2010.

[ZY08] Xianrong Zheng and Yuhong Yan. An efficient syntactic web service composition algorithm based on the planning graph model. In *ICWS*, pages 691–699, 2008. IEEE Computer Society.

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