Atomic service-based scheduling for web services composition

Han, Shukang

Atomic Service-Based Scheduling for Web Services Composition

MSc Thesis (Research)

Shukang Han

Department of Computer Science

University of Durham

September, 2006

Copyright

Acknowledgements

First of all, I would like to thank my supervisor, Dr. William Song, for many insightful conversations during the development of the ideas in lab work, and for helpful comments on research ideas. Also thank my co-supervisor, Prof. Malcolm Munro, who offered many helpful suggestions as well.

My parents have financially supported me and given me a lot of inspiration throughout the research period and I would like to give my special thanks to them.

Finally, I also appreciate the help from colleagues in the department to aid my research work.

LIST OF CONTENTS

LIST OF FIGURES

Abstract

With the rapid development of Internet technologies and widespread of Internet applications, Web Services has become an important research issue of World Wide Web Consortium (W3C). In order to cope with various requirements from service users, services need to be thoroughly and precisely described, thus improvement needs to be made in describing services as more properties should be added to the current service description model based on OWL-S, an ontology structure consisting of service profiles and operations. Semantics is widely considered as one of the core supplements, which is able to provide the metadata of services, so as to better match requirements with services in the service repository.

On the other hand, Web Services has attracted people from various fields to perform relevant experiments on how to cope with users' requirements. Service providers tend to coordinate service implementation by means of interacting with available resources and reconstructing existing service modules. The integration of self-contained software components becomes a key step to meet service demands.

This thesis makes contributions to current service description. The introduction of the term "Atomic Service" is not only considered to be a more refined service structure, but also serves as the fundamental component for all service modules. Based on this, the thesis will discuss issues including composition and scheduling, with the purpose of building interoperations among composable service units and setting up the mechanism of realising business goals with composite services under the guidance of the service scheduling language. This notion is illustrated in a demonstration system to justify the manageable interrelationship between service modules and the way of composition.

Keywords

Web Services, Atomic Service, Description, Composition, Scheduling

Chapter 1 Introduction

The widespread of Internet is capable of converging services and resources at different locations for people to use. The enhancement of the WWW technologies has made numerous services and transactions available on-line and the requirements of these online activities have been increasing at tremendous speed ever since. More and more people are performing transactions using the Internet services and it is a convenient way for their personal information to be managed automatically by online systems.

Companies, especially business corporations, would prefer deals to be operated online rather than in traditional service mode. Compared to the traditional business mechanism, online activities are able to make good use of the Internet for better resource utilisation and service management. The mechanism of realising services, however, needs to be thoroughly analysed at the company side as being the service provider, while it can be totally unknown to service users. Service users, who only care for the quality of the result, do not care much about the internal processes of a requirement. The way a service is dealt with should be managed by service producers. They have to know the details of various requirements, the resources used and the means of presenting results to customers.

The increasing requirements of online services facilitate the proposal and application of Web Services, which aim to provide an interface for using available software components via standard network protocols. In order to ameliorate the quality of services, issues such as service description (how to describe an available service in an appropriate and precise way), service matchup (how to make a user's requirement recognisable by a server), service composition (how to use existing services as fundamental components to be integrated to fulfil a new task), and service scheduling (how to organise pieces of software in a restricted time period) have to be addressed.

1.1 Background

The World Wide Web has been developing in various aspects at tremendous speed since it was invented by Tim Berners-Lee over a decade ago [36]. The boom of the Internet technologies not only facilitates useful information via the web more accessible by users, but also enables the Internet resources to be more interoperable by both service requestors and service providers. As the demand of online transactions increases, it is critical for web applications to cope with various requirements from service users.

1.2 Web Services

Being one of the focusing points raised by W3C (World Wide Web Consortium), the research on Web Services has been carried on throughout the web application area for years. From a service user's points of view, they may raise various kinds of demands and expect them to be fulfilled, while for service providers, the optimal solution is to make user requirements more applicable and realisable by using existing services or with just minor modifications. Therefore, it comes to the problem whether the expectations of service outcome from both sides can be matched properly. It all depends on what their definition of Web Services is.

1.2.1 Overview

The concept of Web Services was proposed several years ago with the purpose of making the applications and services on different platforms interoperable to each other. However, the term Web Services can be confusing for its appearances in many different occasions. Generally speaking, Web Services can be viewed as an application which is able to be invoked by others through its API. Applications can be located distributedly, each of which has its own programming interface to allow requests from other applications or programmes to perform communications, especially exchanging data between them.

Here is an example of a general Web service: a person would like to make a query of the local weather. The query system itself can be viewed as a service. Being a service user, the person needs to submit his requirement to a service registry, which is a repository of all kinds of service information. According to the description from the user, the repository will search for the most matchable service and return the result to the user. Therefore, this weather query service is invoked and executed upon the user requirement and terminated by retrieving the result, which is needed to be checked with the original demand from service users. On the other hand, this weather forecast service, published by providers such as weather stations, can be invoked by other applications as a component, which means the service itself is available to be composed into a more complex service.

A more refined definition regards Web Services as a new platform which facilitates interoperable distributed applications. The UDDI Consortium characterises Web Services as "self-contained, modular business applications that have open, Internet-oriented, standards-based interfaces" [43]. It uses service-oriented architecture (SOA) and intends to offer services through standard web protocols. In a SOA environment, services are made available within the service repository by service producers for access of users to meet their requirements.

According to W3C Working Group Note [41], Web Services is "a software system designed to support interoperable machine-to-machine interaction over a network." In order to implement abstract Web Services, an agent is needed for sending and receiving messages. Therefore, the agent can be a piece of software which acts as a service carrier.

In a word, regardless of different versions of definitions and no matter in which way Web Services is interpreted, the general process of engaging Web Services depends on the interaction between service requestors and service providers based on service

description and message exchanging via appropriate protocols.

1.2.2 Web Services Specifications

In order to provide better understanding and applications of Web Services, the organisations including W3C, Microsoft and IBM have worked through the years, trying to set up some generally-approved standards so as to cover as many aspects of Web Services as possible. Those standards specify the regulations to build Web Services and criteria to evaluate them. Most commonly used standards include Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), and Universal Description, Discovery, and Integration (UDDI), which we are going to describe as follows.

Web Services Description Language (WSDL) [10] is an XML format language published to describe Web Services. WSDL provides services along with SOAP and XML Schema and presents description on communications of services.

Simple Object Access Protocol (SOAP) [19], as mentioned above, is one of the protocols for message exchange via the network. It manages XML-based messages and is widely used in the environment of SOA (Service-Oriented Architecture). SOAP mainly uses the RPC (Remote Procedure Call) pattern to make connections between clients and servers.

The access to a directory is necessary for both service requestor and service provider. The former aims to find exact services to meet their requirements whereas the latter uses the directory as a medium to publish their services. Web Services provides a platform-independent, XML-based registry for businesses by means of nominating Universal, Description, Discovery and Integration (UDDI) as one of its core standards.

Web Services has not only the functional part, which deals with the implementations of fundamental service functions, but also the business part whose task is to fulfil business requirements. For the latter part, a business process language called Business Process Execution Language (BPEL) is designed to define the interactions of business processes.

Moreover, there are standards dealing with other facets where specifications are set up within the context of Web Services. For example, Web Services Resource Framework (WSRF) is proposed on the basis of the close relationship between the execution of services and resources. With the purpose of creating a safe environment for transactions, security specifications such as Web Services Security and Web Services Trust tend to provide a means of applying security for applications in communications [22].

1.2.3 Applications of Web Services

Like other distributed technologies, the most important thing for deploying Web services is to make distributed resources accessible and usable to perform a task.

At present, a lot of companies, such as Amazon.com, eBay, and PayPal provide open public Web services in terms of online transactions because of its availability of collecting all necessary resources to meet user requirements. Customers can use these services to handle transactions such as searching products and making auctions, opening bids and dealing with payments. For service businesses, on the other hand, some companies developed application server software to deploy Web services, for example Websphere by IBM and WebLogic by BEA Systems.

1.3 Semantic Web

Unlike human beings who are able to use the Web to search for something or to do online shopping, it is difficult for computers to do the same thing as they do not

understand the meanings of web pages. This is simply because the current web pages are designed to be human-readable rather than machine-processable. Nevertheless, machines can perform given tasks quite efficiently if all the web information is described in a standard, machine understandable manner. For the purpose of making web pages machine-understandable, W3C has proposed the semantic web as an extension to the current web written in HTML in terms of Internet standards and markup languages.

The proposal of the Semantic Web is to allocate explicit meanings for web information so that machines can automatically process the information [20]. By using the technologies such as XML, Resource Description Framework (RDF) and Web Ontology Language (OWL), the Semantic Web is capable to provide semantic descriptions to the content of web documents.

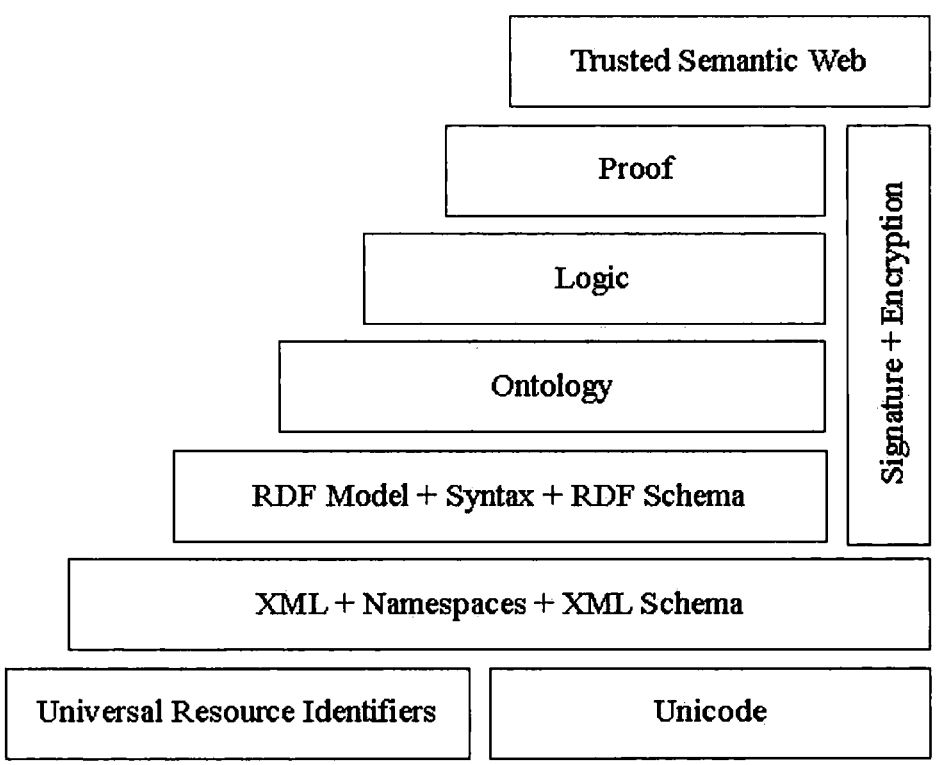


Figure 1-1: W3C Semantic Web Stack

Figure 1-1 illustrates the Semantic Web Stack defined by W3C [5]. It shows that the Semantic Web stack mainly comprises of standards such as XML, XML Schema, RDF, RDF Schema and OWL. Compared to XML which only provides syntax for structured documents and has nothing to do with semantics, RDF defines a data model to describe web resources and their interrelationships. By using RDF Schema to describe RDF resources, semantics is generalised for classes and properties for those resources [13]. In order to facilitate machines' capability of processing information on web pages, the Semantic Web aims to asking people for more effort in providing comprehensible information for machines to understand, such as writing web pages in certain standards in order for machines to extract core information, rather than directly asking machines to master human languages, which has been proved difficult by researchers on artificial intelligence.

1.4 Service Composition and Scheduling

The realisation of user requirements depends on the execution of corresponding services. However, the variety and complexity of requirements result in the impossibility of fulfilling a requirement by invoking just one single service. In most cases, fundamental services with simple functions need to be composed together according to certain integration rules in order to meet complex requirements. The procedure of such kind of composition is not simply to select a collection of services from the service repository. Services have to be scheduled based on appropriate scheduling algorithms to be put together rationally and practically. Furthermore, issues like services with similar functions and occurrence of exceptions during service composition are also needed to be taken into consideration. Therefore, the scheduling on service operations which aims to managing the entire service flow becomes more and more important and has drawn an increasing number of attentions in recent years.

1.5 Criteria of Success

The research in this thesis will address the following issues:

1. Discussions on current service description, composition and scheduling approaches;
2. Proposals of the modified service description structure based on the introduction of the concept of Atomic Service;
3. Introduction of the service composition and scheduling issues based on Atomic Service;
4. Illustration of the Atomic Service-based scheduling and composition through the implementation of a demonstration system;
5. Reasoning and evaluations on service structure and approaches of service composition and scheduling.

1.6 The Structure of Thesis

Through reviewing current research work in related areas, this thesis covers semantic-based Web Services architecture, description, composition and scheduling, while the issues of service composition and service scheduling are based on the descriptive model of Web Services.

The rest of thesis is arranged as follows:

In Chapter 2 Literature Survey, a deep study on Web Services related fields in terms of service structure and description, service composition and matchmaking and business process, etc. is performed before drawing the conclusion of the necessity of making amendments to the existing architectures and proposing new methods of dealing with service transactions.

In Chapter 3 Atomic Service-Based Scheduling, by putting forward the modified model of service architecture, the concept of atomic service and its properties is addressed followed by the discussion on service composition and scheduling issues.

Chapter 4 System Design and Implementation: a demonstration system based on the

proposition of atomic service to illustrate the solutions to service composition and scheduling is introduced analyses on both functional and business end of Web Services.

In Chapter 5 Reasoning and Evaluation, the discussion on the approach to better fulfil a service requirement based on the proposed concepts and methods with the help of giving an example on classes is put forward. After that, the evaluation of the atomic service-based scheduling method is made by means of performance comparison with other scheduling approaches.

Finally, in Chapter 6 Conclusions, the conclusion of the thesis is made through the discussion on both the modification of service description and the proposition of atomic-service based service scheduling, with some open problems and future work being presented afterwards.

Chapter 2 Literature Survey

2.1 Overview and Structure

Web Services is an important issue in the domain of internet applications, which raises much attention not only from service producers, but from non-specialists as service users as well. Generally speaking, Web Services aims to collect all available services throughout the web and utilise various possible resources to meet all kinds of requirements. In terms of the relationship between service producers, services users and the directories of services, researchers have been keen to seek a way to best match their relationship and make certain improvement to existing web technologies.

In recent years, there are several important issues in the scope of Web Services which people put more focus on. One of the most important aspects among them is how to match the requirements between service users and providers. This is because demands raised by service users are usually expressed in natural language and the majority of those users are probably non-specialists in the area of Web Services. Therefore, it could be possible that their requirements are quite vague and not that matchable to those existing descriptions of services. This, however, could easily lead to failure of finding the exact service they need. It is not difficult to think about adding semantics to the description of services to solve this problem. Moreover, the existing structure of service should somehow be modified in order to best fit the possible requirement.

Like software reuse, it is impractical and unnecessary to start with the fundamental components of services or functions every time when a new requirement is raised. It may, to some extent, contain some existing mature functions that could be used directly or just with some small refinement. This process will save a lot of time and reduce the possibility of making errors during implementing those basic functions. This can be somehow described as "Software as a Service".

In the point view of service users, they would not only like to get their requirements to be implemented, but also like them to be processed correctly and on time. For this reason, the service scheduling mechanism should be taken in consideration during composition process. Different ways of integrating fundamental services result in complex services with various functions and a service may possibly have an alternative service with similar functions. All of these aspects require a certain configuration to monitor and manipulate the whole process of composition.

Moreover, there are still lots of hot point problems apart from the above-mentioned aspects in Web Services. Therefore, this rest of this section will go over some related key research work in recent years and draw some corresponding problems with respect to the existing services structure, description, modelling, operations (composition and scheduling), and etc.

2.2 Web Services Definition and Structure

2.2.1 Formal Definitions of Web Services

The World Wide Web Consortium (W3C) defines Web Services as the programmatic interfaces which make application to application communication available [48]. It can be inferred from this definition that Web Services focuses on transactions of applications, thus also known as service-oriented architecture (SOA). However, there are no unified definitions of Web Services up to now.

According to Oasis, Web Services is defined as a software component described by WSDL. It can be accessed by standard network protocols such as SOAP over HTTP [8]. Meanwhile, as specified in Web Services Architecture Requirements, "A Web Service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web Services in a manner prescribed by its definition, using XML based messages conveyed by Internet

protocols." [3] Although there are no agreements on specifying what Web Services exactly is and its definitions have always been under hot debate within the W3C Web Services Architecture Working Group, it is commonly acknowledged that Web Services is a term to describe an entire breed of applications, referring to technologies which allow making connections among them.

Generally speaking, Web Services consists of two main aspects: business service and functional service. The business part of a service, which deals with the producer-user interactions, focuses on the application of service. The functional properties of a service, however, are mainly concerned of the mechanism of realising a service in a concrete way.

2.2.2 Service-Oriented Architecture

After it was first described by Gartner in 1996 [45], the Service-Oriented Architecture has been widely regarded as the evolution of both object-oriented design and distributed systems by building up the use of services to support software user requirements. With the emergence of Web Services, some people take it for granted that SOA can be specifically referred to Web Services. However, not all SOA is based on Web Services and Web Services is not interpreted to SOA in every case. Nevertheless, the relationship between these two technologies is influential as the architecture of SOA makes the operation of Web Services successfully [33].

Although it is the fact that SOA does direct a way to apply new technologies such as SOAP, WSDL, and UDDI within real business scenarios, tactical and implementation mistakes in using SOA will inevitably produce disturbing programs. There are many messaging protocols which can be used for SOA in addition to SOAP over HTTP for services. Therefore, it is critical to use the right or a mixture of protocols for appropriate services. Moreover, services can never be regarded as collections of reusing codes. Regardless of numerous ways to reuse code and functionality, good

services should also concern about resource use.

Based on the structure of SOA, the systematic architecture and fundamental components of Web Services are depicted as in Figure 2-1.

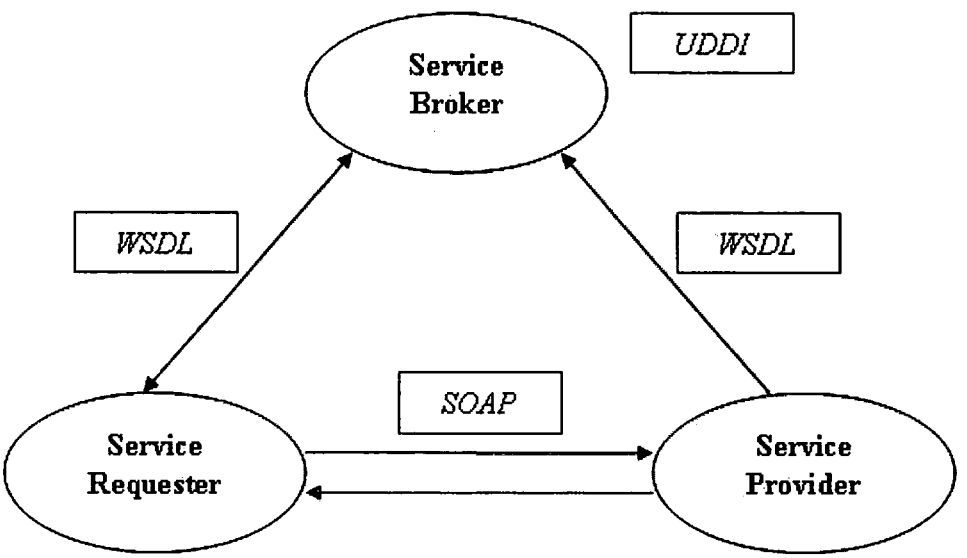


Figure 2-1: Structure of SOA

It is illustrated from the graph that SOA mainly consists of three parts: service provider, service broker and service requester, each of which makes its own contribution to the whole architecture.

(1) ***Service Provider:*** to publish its service and give response to any request of using its service;
(2) ***Service Broker:*** to register and classify service providers which publish services and use Universal Description, Discovery, and Integration (UDDI) as the registry to enable applications look for services to determine whether to use them;
(3) ***Service Requester:*** to search for services by means of Service Broker and make use of appropriate services afterwards.

Therefore, there are three operations among these entities to guarantee any process

over the architecture above:

(1) ***Publish:*** the operation imposed by service provider on service broker to register both functionalities and accessible interfaces of services with the help of Web Services Description Language (WSDL);
(2) ***Find:*** service requester uses this operation to look up appropriate services according to descriptions based on WSDL;
(3) ***Bind:*** the operation which facilitates service requester to interact with service provider with the aid of Simple Object Access Protocol (SOAP) and use its services.

The emergence of Web Services is more or less based on the following two mechanisms. First, protocols such as HTTP and TCP/IP, as well as XML standards and SOAP, have made communication more pervasive across the internet. Furthermore, structures brought forward in light of UDDI and WSDL have made services described, published and usable. However, with the boom of service demands in various scenarios, more refined description of needs to be attached as additional properties for services, which results in the emergence of service-bound metadata.

According to Cabrera et al [9] and W3C [7], the architecture of Web Services at least contains components as follows: messaging, discovery, resources, semantics and security. It is depicted as Figure 2-2.

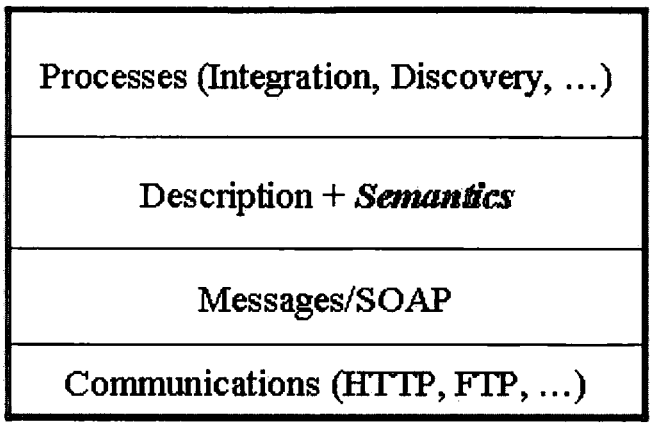


Figure 2-2: Web Services Architecture

However, there is no standard specification to define what component Web Services exactly contain. But mostly admittedly, these are the core parts to constitute Web Services:

Messaging: a core part which deals with data transmitting between Web Services agents. It is a channel to pass information through models and makes communication happen both inter-services and intra-services.

Discovery: using registries to locate services and performing availability check. It includes matching services according to their descriptions.

Resources: any entity which has an identifier.

Semantics: using metadata in service description in order to make machine understand

Security: a mechanism based on XML to secure safety issues such as message integrity and confidentiality.

2.2.3 WSDL

Web Services Description Language (WSDL), as mentioned in Chapter 1, is the language based on XML to describe and locate Web Services and to specify the

methods which are used to get access to services [10]. The main structure of WSDL and the major elements used to describe a service is as follows:

```
<definitions>

<types>
    definition of the data types.....
</types>

<message>
    definition of the messages....
</message>

<portType>
    definition of the operations.....
</portType>

<binding>
    definition of the communication protocols....
</binding>

</definitions>
```

From the syntax involved in the WSDL document above, we are able to understand the four elements which are grouped together to describe a service.

Types element: declares objects relating to a system which processes the WSDL document.

Messages element: describes messages which are sent between the client and the

server.

Port Types element: identifies operations and messages.

Binding element: specifies protocol details for operations and describes the content of messages.

Although WSDL is widely used to describe services running on any protocols, as it provides all necessary information to establish the communication with a SOAP server against a client. However, WSDL is not enough to reflect the interaction flow between business parties as it lacks lots of properties [46]. It is because SOAP transaction, on which WSDL is erected, involves only document exchange rather than multiple transactions. Therefore, other approaches need to be considered to specify business flow arrangement.

2.3 Markup Ontology Languages and Service Modelling

Given a service, it is important to understand the ontology, its hierarchical structure, before adopting an approach to deal with it. There are several markup languages to represent ontology and encode knowledge.

2.3.1 DAML+OIL

DAML+OIL aims to describe the ontology infrastructure in semantic web to facilitate the web to understand the meanings of web contents and perform reasoning with the information, rather than just displaying them. It is also widely used to describe ontologies of various domains, and a major evidence of importing biology into DAML+OIL can be found in the way of describing gene technology [14].

Built on top of DAML+OIL, DAML-S is a semantic markup language to describe service and ontology. Unfortunately, Klein et al [25] think that current DAML-S is not sufficient enough to describe existing services as it does not trace the step-by-step service execution and thus, is unable to support stepwise service refinement. As

discussed, it is possible to successfully reinforce the incomplete service descriptions by using variables. This is because that current state/message mechanism is not able to describe more interactive services rather than fixed ones. However, the concept of variables is not provided in either DAML-S or dependent standards. Therefore, a structure to add variables to DAML-S based service descriptions is proposed. Inferior to other standards, the modified service description is not able to automatically generate GUIs.

2.3.2 RDF

Based on URI (Uniform Resource Identifier), Resource Description Framework is designed to represent information about resources, especially metadata about web resources [26]. A simple RDF statement is shown in Figure 2-3.

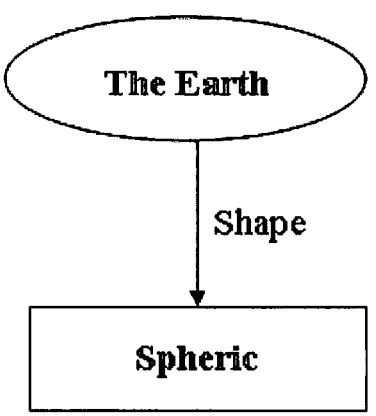


Figure 2-3: A Simple RDF Statement

From Figure 2-3, it can be seen that the individual the earth has the shape property which is spheric. This statement can also be represented in RDF/XML format as follows:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:description="http://.../planet#">
        <rdf:Information rdf:about=" http://.../ planet#earth">
```

```
                <description:shape>Spheric</description:shape>
</rdf:Information>
</rdf:RDF>
```

Comparing with the code segment with the graph, we are able to find out that the resources, planet and earth, are both defined and the relationship between the earth and its property, spheric, is also specified.

Nowadays, it is still a difficult problem to automatically generate service description, while it is impossible to manually produce the description. With RDF, it is feasible to generate metadata information automatically for discovering services [23]. The metadata generator can then link RDF description with a web page and the metadata element specified by RDF can fit in a search engine for service discovery.

2.3.3 OWL-S

Built on top of OWL, OWL-S makes it possible for service discovery, execution and composition. For example, problems remaining in these services issues may include: In what way can the constraints be satisfied during searching and discovering a service? How to automatically invoke services which are normally performed by humans, such as filling forms during a purchase procedure? And how to let machines deal with a travel arrangement which composes and interoperates several itineraries by just offering some prerequisites? In order to deal with these kinds of questions, OWL-S tries to define the meaning of what Web Services can offer and what information is required. According to Martin et al from W3C [27], the upper ontology for Web Services created via OWL-S can be shown in Figure 2-4.

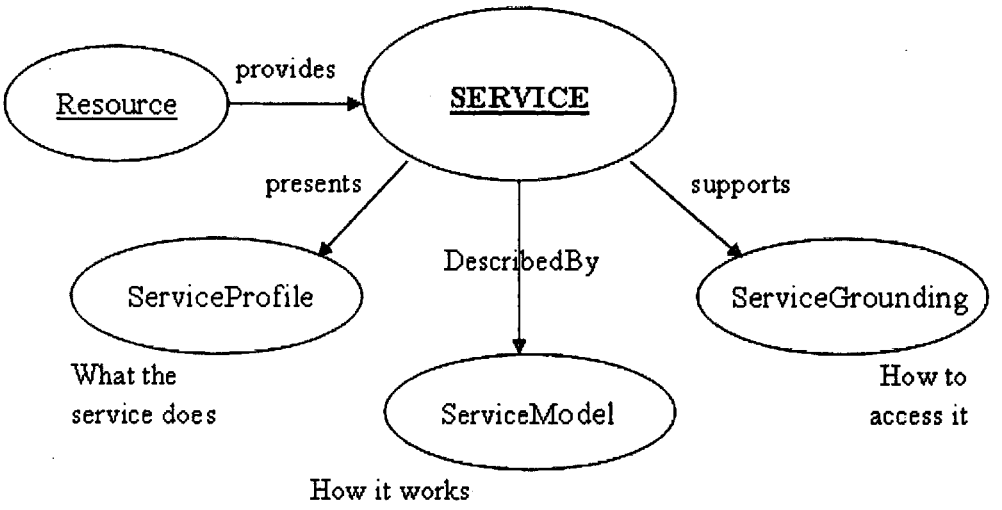


Figure 2-4: An Upper Ontology for Web Services Defined by OWL-S

As illustrated in Figure 2-4, a service operates based on the provision of resource. The service class has three properties to support three subclasses respectively, each of which characterises the functionalities.

Service Profile: presented by service to express what the service does, especially the requirements from users and the expectations. It also provides the information about service features and enables automatic discovery.

Service Model: describes the service to show how it works, particularly shows the data and control flow of service processes.

Service Grounding: supported by the service to indicate in what way the service is accessed and used. It provides a concrete description of binding protocols and messages, etc.

One of the advantages of using OWL-S is that it is very universal and is suitable for describing any Web Services, as it is designed to let service definition precisely refer to a unified set of documents which describe the exact meaning of service. On the contrary, this characteristic can also be viewed as a disadvantage of it. Because it can be too general that when it is adopted for description, OWL-S has to be extended to

specify more classes and add more properties in order to enable semantic descriptions, as mentioned in [21] for a digital preservation system.

On the other hand, there is also a proposition to apply UML class diagrams to define schemas, which are part of the service ontology [38]. However, ontology has a restriction that an object must have one of the characteristics of a class in order to be an instance of it. As a result, compared to OWL-S, the UML approach is not suitable to define ontology although it can be bound with other techniques to describe services issues.

2.3.4 Service Modeling

Current service models derived from languages such as DAML-S, RDF and OWL-S are not sufficient to describe various kinds of service requirements. In order to enhance the universality of them to cover service descriptions, a lot of researches make use of the expandability, such as to import supplementary properties or functions to model services [25, 21]. Another example is illustrated by Chung et al [11]. Based on OWL specification, they have proposed Service-Oriented Process Models to build services to support collaborative design and manufacturing. The framework, which has distributed architecture, is able to cope with heterogeneous networks. Moreover, de Bruijn et al [12] have addressed that Web Services Modeling Ontology (WSMO), which specifies the ontological structure and descriptions for the core elements of semantic Web Services, can be used to enable business integrations based on formal descriptions.

On the other hand, Web Services models which are not based on those languages have also been proposed to handle other related issues. Currently, there is no support for comparing the ratings of service standards which enable service publishing and discovering. Therefore, a conceptual model for service reputation has been designed by Maximilien and Singh [29] in order for generating a new approach for service

location and selection.

2.4 Service Composition and Matchmaking

A service requirement corresponds to a task which can be realised in the point view of fulfilling its composite tasks. The procedure of dissecting a complex task into fragments follows the aim of refinement, resulting in implementable components. On the other hand, given a repository of executable fundamental services, the key issue is to seek a way to integrate appropriate ones into a compound with various functionalities. This involves the topic of how to better match existing services to given demands.

2.4.1 Software Reusability

Web Services, in some ways, is regarded as a reconstruction of several existing pieces of components which are integrated to make complex performances. In this case, the mechanism of service composition overlaps with the concept of software reuse, the issue which was put forward probably over three decades ago.

Software reuse refers to the notion of making partial of complete computer program code reusable for a later project in order to reduce both software development timescales and costs. The purpose is to avoid repetitious work on same applications in the case of multi-use of identical or similar pieces of software components. Compared to constructing a program from those very fundamental functions or methods in lib files, it is much easier and efficient to search for and adopt existing software components to form a new application.

Although it aims for reducing time and cost, pure software reuse has been proved to be unsuccessful for the past mainly because of the following two reasons [40]. Firstly, reusable components are abstract and mostly encapsulated as a whole integrity, which disables any amendments to be made and the desired outcome, cost and time may be

affected. These all make it difficult to analyse and evaluate their quality before adopting them. Secondly, the skills used to support reusable components tend to become unconvinced, which also makes it almost impossible to trace software reuse.

Nevertheless, people have been seeking ways to make software reuse feasible, especially when some of its notions can be shared with the proposal of Web Services composition. For example, the Case-Based Reasoning technology (CBR) is introduced by Gu for component recognition and retrieve [18]. According to the purpose of software reuse, one of the key issues is to discover appropriate software component and apply it to wherever required. However, with the dramatic increase in the number of available software components, it is far more than an easy task to locate proper components. Moreover, given numerous components, it is inefficient to go over all of them to find the target one. In this case, a knowledge-based component retrieve model was proposed in order to explore appropriate component while not imposing excessive work on service queries. Most importantly, the CBR mechanism offers a way to represent both general domain knowledge and component knowledge, reflecting general quires and component providers respectively, and uses both of them in component retrieval process. It can, therefore, effectively reduce the conflict brought by different representation of the requirements and corresponding components.

2.4.2 Composition

It has been a growing trend with the developing software architecture to combine multiple applications in the distributed environment. The build-up components are functions extracted from several different resources, maybe individual and partial functions, or even an entire encapsulated system. The composition issue, then, has drawn a lot of interest, especially for business and enterprise application integration [44].

According to Paulo F. Pires et al [35], Web Services composition is "the ability of one business to provide value-added services to its customers through composition of basic Web Services, possibly offered by different companies." From this definition, it is suggested that there are at least two fundamental attributes of those "composite services". First, they are integrated from basic services and should thus inherit the properties of basic services and be compatible to those APIs provided by them. Second, composite services are not only a simple collection of basic services. They should be "value-added service", which means that composite services are reckoned to have extra functions and these functions cannot be realised by those individual basic services.

Various methods which are keen to facilitate service composition to be more comprehensible and easy-executable have been carrying out throughout these years. One of them is prone to being applied by using BPEL, Business Process Execution Language at business level, and it is called Business Process Execution Language for Web Services (BPEL4WS) when used within the scope of Web Services. It has superseded Web Services Flow Language, an XML language proposed by IBM to describe composition formerly. BPEL provides a language for specifying business processes and business interactions and defines a model to facilitate process integrations of business services which will be discussed later.

In order to make elementary services be easily composed, Mostefaoui and Hirsbrunner [32] believe that it will pave the way for service composition by using the composition framework based on context. They bring in a conceptual framework of Context-Based Service Composition consisting of four layers, each of which has certain functions to implement within its own processing region. In the authors' point of view, the purpose of introducing context is to make services more discoverable, easy to be composed and executable. However, the proposed composition model does not help service composition to be executed in an automatic way, which still requires manual manipulation and lacks of consistency in a large scale. On the other hand,

Jianghai Rao et al [39] have addressed that applying linear logic into service composition issue can facilitate automation. Their proposal bases on the assumption that core services which have already been selected by users cannot completely accomplish their requirements. Therefore, the invocation of combinations of core services becomes necessary to fulfil the goal. Contrasted to using languages such as WSDL as external representation to describe services, it is proposed that linear logic can be adopted to internally depict service composition issues. In this case, the composition of services can be converted to the computation and reasoning supported by linear logic, although it is not originally considered as a proper tool to deal with composition issues because of its complexity. Even if it can somehow make an impact on guaranteeing the correctness and completeness of composite services, it cannot intuitionally represent the service flow upon composition and will constrain the freedom of selection among fundamental services. This can more or less hamper the service expansion to a large scale. In order to solve this problem, the notion of using graphs to represent service and its interactions has been proposed with the introduction of conceptual graph and other graph-related representations. The graph-based service issues will be discussed later in this thesis.

According to Matskin and Rao [28], the infrastructure of Web Services composition is depicted as in Figure 2-5. The service composer plays a vital role in the whole composition process. First of all, it extracts key information from the description of a require service. The composer then starts both of its components, workflow manager and component integrator. The latter one directly interacts with UDDI, trying to find a matchable service profile. The optimal service which exactly matches the specified requirement, if existing, will be selected. Otherwise, the component integrated needs to start a composition process to synthesis available services according to their specifications. The workflow manager monitors the whole process and models the workflow by using workflow language or service flow language. The two components of the service composer both contribute to forming the composite service.

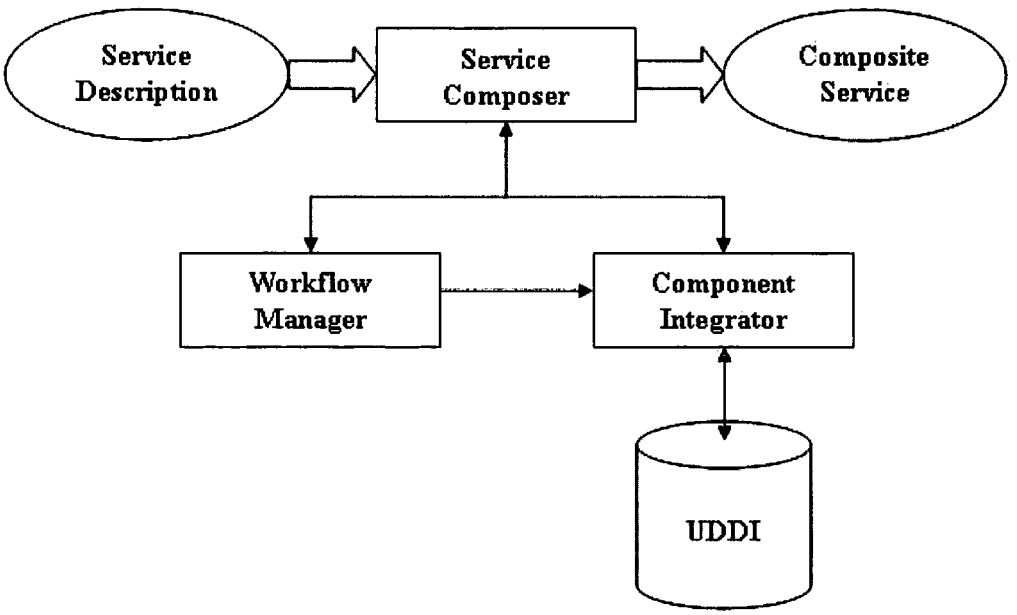


Figure 2-5: The Infrastructure of Web Services Composition

Web Services, on the other hand, focuses on business services which also include issues such as integration and composition. Similar to service composition at functional level, the selection of distributed services from different service providers is a critical problem in business processes. Taking factors such as time and cost into consideration, a service can only be appropriate to be selected from service pool if it can fulfil part of a business goal or is an indispensable component in the composite service flow. However, the selection of service providers is far from a simple task because of the amount of available services and complexities among service correlations, as the implementation of one service can probably make significant impact on the choice of related services. In order to provide an efficient way to allow customers to select and obtain optimal services, Roman Ginis et al [15] have proposed a method for modelling business processes through service composition. It is suggested that service requestors have to formally express the activities and relationships of business processes they are in need for, and thus automatic tools can help them to make selections and reservations optimally. However, regardless of the difficulty in deploying such automated tools, it is not feasible to expect any "formal

expressions" from service requestors. Those requirements, in normal cases, are described in natural languages and service requestors, if not specialists, are not probable to offer requirements in those formalities. Provided by those mere human-understandable demands, it is impossible for machines to match them with the existing patterns. The solution can be made through improvement on the description of services by adding semantics. This is to make services well described and possible for machines to matchmake service requests with appropriate ones in the service pool.

2.4.3 Current Composition Approaches

2.4.3.1 BPEL

Since the first-generation composition language Web Services Flow Language was superseded because of its incompatibility, researchers have been carrying out several approaches for service composition, represented by BPEL mentioned above. Although there is no standard by W3C for composition at this stage, those proposals have already made contributions to composition in some ways.

Among those composition methods, BPEL4WS is the major approach for service composition and is the standard by OASIS (the Organisation for the Advancement of Structured Information Standards). In order to get the process as the composition result in BPEL, activity exchanges messages and processes are able to interact with other participating services via WSDL [31].

The approach of using BPEL to perform composition is based on a framework of two patterns: workflow pattern and communication pattern [50]. In the workflow pattern, the control-flow, based on the workflow model, monitors the composition procedure and the flowchart-like BPEL process specification make it practical to represent complex structures of composition. Moreover, because BPEL is a communication language, the communication pattern facilitates that all the activities within composition can be completed via sending and receiving messages. However, one of

the major deficiencies of BPEL is the lack of semantics. The unclear expression of semantics makes it difficult in interrelating services for composition. Therefore, relevant logic and reasoning need to be considered in order to be mapped into formal representation to create and correlate processes.

On the other hand, since BPEL is proposed for describing business processes, it tends to contribute to the outcome associate within a business activity and improve the performance of the process. It can make business process interact with external entities through WSDL, thus capable of facilitating automated process integration. However, from the organisation perspective, BPEL does not give a proper method for measuring the performance of service execution. First, BPEL can only manage automatic activities and is not able to handle non-automatic business processes. Therefore, it is not capable to manage all sorts of business processes and inefficient to deliver any kind of business process descriptions from one to another. Moreover, the immaturity of BPEL leaves lots of gaps in its specification, which makes individuals who would like to use BPEL need to set up their own implementation ways to fill in those gaps. Several proposals of extending BPEL with essential properties have been put forward. For example, in [30], a method for extending BPEL in order to measure business performance is proposed. It applies Solution Management to the current Web Services architecture, providing three management type categories of define, log and analyse. This definition of service, which is different from the description offered by UDDI, includes business level information, especially the performance measurement information, while the log service provides a mechanism that the data transmission in the workflow can be monitored. At last, the analyse service is joint for making request of existing services and analysis of process performance according to the defined criteria.

Besides BPEL4WS, Semantic Web, precisely OWL-S, can also make an effort in service composition as provides the standard of enterprise integration and sharing data, thus enabling service discovery, invocation, composition and interoperation. The

process model in OWL-S provides service descriptions in terms of input, out, precondition and postcondition, etc., all of which are prerequisites for service composition. One of the other important composition methods is based on web components, specifically reusable pieces of software, whose interface enables discovery and reuse.

Because of its lack of semantics support, BPEL cannot let service users define some non-functional properties such as the quality of service, while OWL-S is able to manage it. However, neither of these two approaches can guarantee the correctness of composition while web component method succeeds in doing this with the help of composition logic.

2.4.3.2 Service Composition Using Conceptual Graph

Conceptual graph is a logic system based on the existential graph and the semantic network of artificial intelligence. It provides an abstract expression by using nodes to represent concepts and conceptual relations and arcs for liking them together [42]. The linear form of representation is as follows:

[Graph: {x}] → (Attr) → [Conceptual]

This denotes that graph has an attribute which is conceptual, which means a conceptual graph. The two concepts, graph and conceptual, are linked together by the conceptual relation of attribute.

Because it has the advantages of both human-understandable and machine tractable, it is able to interpret between computer-based formalities and natural languages. Moreover, conceptual graph can be used to represent describe service and service composition. We will use the two processes, List and Pick, in the online purchase example above to illustrate.

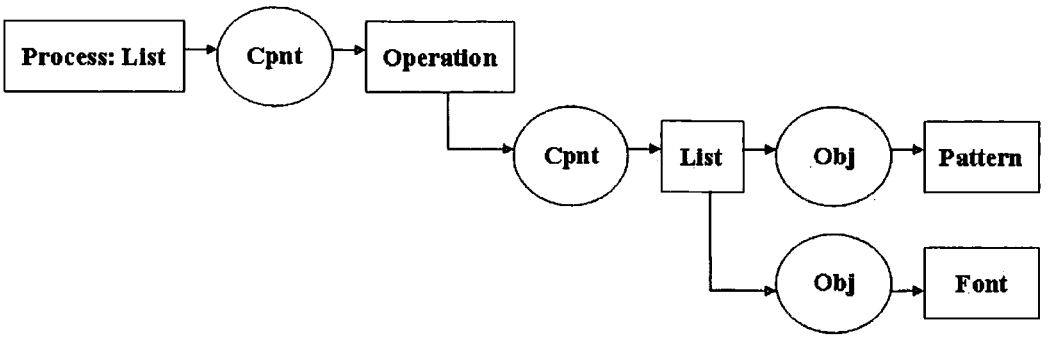


Figure 2-6: The List Process in Conceptual Graph

Figure 2-6 depicts the list process in conceptual graph. The process, which has the name of List, is composed of several operations. Therefore, the relationship between the process and operation is composition and operation is the component of the process List, denoted as Cpnt. There all numerous of operations available and list is the one that should be selected here. Thus the relationship between operation and the action list is also composition. It should be mentioned that the meanings of the two Lists are different. The first one denotes to the name of the process while the second list refers to the exact operation. In order to perform the list operation, both the list pattern and the font in display needs to be considered, which result in the two objects of list: pattern and font.

Another process Pick to be expressed in conceptual graph is shown in Figure 2-7.

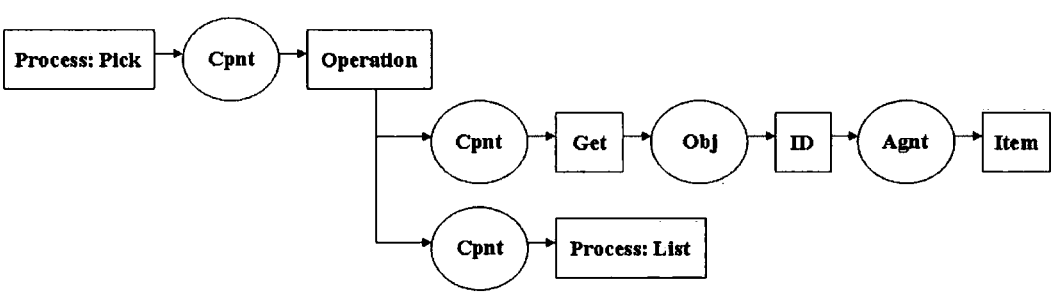


Figure 2-7: The Pick Process in Conceptual Graph

Like the list process, the pick process has component which is operation. There are two operations to be selected to implement this process. The first one is the action of

get, which has the object of the ID of the target item. The other operation is the reproduction of the list process, which means the whole pick process needs to be composed with list to be implemented.

2.4.4 Matchmaking

As mentioned above, despite the tremendous progress towards the on-demand issues in Web Services, it is still far from perfection in matchmaking existing services with various service demands. In order to implement a proposed requirement, the most important thing is to search for an appropriate service in the service pool, either seamless matchable or at least with similar functionalities that can almost fulfil identical requirements. UDDI registry, in terms of the scope of Web Services, stands as the repository of services which shares an analogous performance as a service pool with search functions. The service selection issue now comes to the point of seeking matchable components in the registry. However, the search mechanism for traditional UDDI business registries is more or less based on key-words, which gradually proves to be inefficient and impractical. For one reason, it can be hard for service requestors to extract proper keywords from their requirements, especially when the requirements are vague as the requestors themselves are not capable of making clear expressions of their demands. This will lead to numerous redundant return services which bring additional time and cost. Like search mechanism used in lots of current search engines, the most important thing in order to successfully meet the expectations is to provider keywords as appropriate as possible to locate the search goal. However, in the service users' perspective, it is not a pragmatic solution to provide all precise description of their requirements to exactly match the description of existing services offered by the service registry. Therefore, it is necessary to bring in a proper method of matching different description from the two sides to pave the way for service search and discovery.

UDDI is a standard service registry, as well as a search engine. However, as

mentioned above, its functionality of being a search engine is limited because it is only based on keyword retrieve. In order to enhance its ability to discover services, Kawamura et al [24] have performed an experiment on applying semantic service matchmaker with UDDI. By combining search with Web Services Semantic Profile, a semantic service description of semantic service matchmaker, it is able to level up the performance and functionality of the matchmaker.

In fact, semantic service matching can not only be realised with the aid of UDDI-based approach, service descriptions depending on other languages can also be applied for semantic service. A matching engine based on DAML-S is applied in the architecture of the DAML-S/UDDI matchmaker [34]. Web based DAML ontology is able to supply the gap of UDDI to perform semantic search.

It can be inferred from above that it is not sufficient to enable semantic search by purely using UDDI. It needs to be combined with a good-formed and semantic-based ontology to support description for semantic search.

2.5 Service Scheduling

The composition of services intends to collect all the relevant and composable services together and integrate them into a more complex service with greater power and functions. During the procedure of composition, an issue has been raised in terms of selecting similar services and organising the sequence of service operations, as composition rules only deal with integrating two or more services together without a bird's-eye view on the overall integration process. Therefore, scheduling mechanism has been proposed in order to solve the problem of service arrangements.

2.5.1 Job Scheduling

The concept of scheduling in Web Services refers to the mechanism of ensuring reliable system processing in terms of monitoring and controlling the service

operations. Service requestors at the user end can ask for a desired service to be put into the operation chain. On the other hand, the service scheduler, which keeps a record of the service selection and execution sequence according to the requirement and decomposition, manages the whole service procedure. For example, according to OMII-Europe, users can submit job requests to the job execution and thus a job scheduler is needed to coordinate the sequence of operation. Therefore, Globus Toolkit 4 [16] provides a mechanism of selecting, controlling and monitoring the workload of jobs to a manageable order.

2.5.2 Scheduling Algorithms

In order to manage the operation sequence of services, Yifei Wang et al [49] introduces several scheduling algorithms that are applicable in different circumstances. The target of scheduling algorithms is to make a balance among services which will be invoked and to avoid both starvation and over-action of a service. There are mainly two styles of scheduling: sequence and priority.

Sequence scheduling, also known as First Come First Served (FCFS) or First In First Out (FIFO), applies to services forming in a queue and needing to be executed in a chronological order. The importance of services in this theory only refers to the direct following service rather than the service which could actually be "crucial" within the whole service flow.

On the other hand, if the result of a particular service is urgently needed or plays as the base of all subsequential services, the priority scheduling algorithm needs to be applied for this service. This includes algorithms such as Earliest Deadline First scheduling (EDF), Shortest Job Next (SJN) and Shortest Remaining Time (SRT) etc. Therefore, it all relies on the different occasions that appropriate service scheduling algorithms need to come into operation.

2.6 Summary

This chapter has presented the reviewed work on related researches mainly in Web Services structure, description, composition and matching. Although there is no current standard definition of Web Services, it is general accepted to be based on SOA structure. There are also lots of languages for describing services and related issues so as to facilitate using services through operations like composition and matching.

There are a lot of current technologies which have close relationship with Web Services or can be incorporated with Web Services for complex performance. For example, Microsoft's .NET and Sun's J2EE can both be used as the application frameworks for Web Services.

Chapter 3 Atomic Service-Based Scheduling

3.1 Introduction

With the rapid development and wide spread of the Internet, the demands of on-line services are growing at a tremendous speed and the growth rate of requirements will by no means decrease in the long term. Those requirements, in terms of the way of expression, are mostly in natural languages which are commonly used in routine life. This is simply because most service requestors are not specialists in this particular area. Therefore, they do not know in what way their requirements can be fulfilled.

These natural language-based requirements, however, are only human-readable rather than machine-processable. It is difficult for machines to extract useful information from natural languages in order to process various requirements. Therefore, service providers, at this stage, have to analyse these requirements and put them into implementation.

Normally, those requirements proposed by service users are quite general and are not always expressed in a well-formed structure. It is because these requestors are not very clear enough with their requests and whether they will be satisfied by the results given. In this case, service providers usually decompose the requested service into several sub-services. This decomposition procedure can be repeated several times until the stage that services cannot be further decomposed when the decomposition has generated basic services that can be realised directly by invoking corresponding functions and methods in the lib file. Services at the bottom of the decomposition chain form a service pool, which contains all the fundamental services and basic components to be integrated. Our research works focuses on the structure and description of these services, which we call atomic services, and how they interact with other services to perform complex tasks.

3.2 Key Definitions

First of all, we will discuss some key concepts in the context of service description and service decomposition. W3C (The World Wide Web Consortium) refers to the programmatic interfaces made available for application to application communication as Web Services. Therefore, it implicates that services can be viewed as "semi-manufactured components" which can be put into practice only after some further processes. In our discussion, we propose the concept of atomic service, which can realise some basic function and furthermore, serves as fundamental component of complex services. In order to realise complex services, atomic services need to be composed together according to some integration rules, taking scheduling issue into consideration.

Sub-service: A service which realises multi-functionalities can be decomposed to "small" services, each of which can realise part of the whole requirement. Although these services, called sub-services, are not as complex as services of high level, they have well-defined descriptions according to our service structure. Therefore, the implementation of the general service relies on the completion of its sub-services.

Decomposition: the procedure of "dissecting" a service into its sub-services with the aim of being easily processed. A general requirement which is represented in natural language cannot be recognised and processed by machines, and it needs to be thoroughly analysed and separated into several hierarchical inter-related components to perform complex tasks.

Atomic Service: a simple service with a lifetime of a single request.
After decomposing a general service to a certain stage, some decomposed services can be somehow directly executable by calling those fundamental functions in the lib file. In this case, it is not necessary to decompose these services any further down to their sub-services (whether they have any sub-services or not) because these executable

services already serve as basic components of a general requirement. These services, denoted by leaf-nodes in the service decomposition tree, are called atomic services, the features of which will be discussed later.

Composite service: Services which are not atomic are composite services, which mean that they are not executable directly from functions or methods within the lib file. The topmost service, which corresponds to the general requirement, is a composite service which integrates all its sub-services.

Composition: the task of putting together atomic and/or composite services to perform complex tasks.

In the circumstance of the disability in realising a required functionality using existing services, it is a good idea to combing existing services together to fulfil the request, in other words, to let the composition be dynamic. This process of composition is essential to both business-to-business and business-to-customer applications especially when Web Services become more prevalent nowadays. Basic Web services will be possible to bear the ability of being value-added through composing with other services provided by different companies [35].

Integration: to combine a set of applications by using some architectural principles, especially used in enterprise application integration by joining relevant middleware components together

Web Services (in terms of Atomic Service): a collection of atomic services, composite services or a combination of both categories.

3.3 Service Decomposition and Composition

Decomposition and composition can be referred to a "Divide-and-Conquer" problem

in the domain of Web Services. For various kinds of service requirement, it is impractical and impossible to find an exact corresponding collection of service on every occasion. In most cases, service requirement does need to be thoroughly examined and decomposed and the realisation of the original service owes to implementation of each of its sub-services. Parameters of the requirement pass down to its sub-services and the result is presented after several operations being completed on sub-service level. This data transmission procedure is opaque to service requestors as they do not need to know how their requirements are processed. However, the decomposition process should be transparent to service providers, as they are responsible for composing functional services and use the performance of compound services to meet the original requirement. Therefore, the whole process can be visualised as a combination of separating a general service into pieces and integrating those completed and useful components back into the original service.

Requirements proposed by service users are mostly in natural languages and are not machine-processable. Service providers have to manually perform the analysis work upon each request. A request, in most cases, can be quite implicit, which means that it is not expressed in a very clear way and can contain several possible meanings. For example, the meaning of the request "Please buy a ticket for me" is vague because it omits lots of useful information. In order for machines to realise the request, service providers will have to convert the natural language to programming language or machine language during the process of decomposition.

First of all, words which play an auxiliary part and do not contribute much to the meaning of the whole request should be removed, leaving notional words (words which have full lexical meaning in context) only. In this example, the words "buy" and "ticket" will remain after the elimination. However, the expression "to 'BUY' a 'TICKET'" still does not give enough information to enable the process of the request as both words have some implicit meanings. In other words, both the action "BUY" and the object "TICKET" can be further decomposed.

3.3.1 Top-down Convention

Given a task, the necessary way to accomplish it is to make a full examination before setting out to draw solutions. A given task can only be implemented after being analysed, refined and thoroughly decomposed to the stage of executable functions. This procedure, which requires a series of decomposition, either manually or automatically, is called top-down convention in terms of service decomposition.

As illustrated in Figure 3-1, if we regard a task as a topmost service, it should then correspond to the initial requirements from service users. Expressed in natural language, a task can be converted to a corresponding service which is processed with the final purpose of being recognised by machines. Services users, if not specialists in Web Services area, may not understand the way of expression of their requests after this step because what they concern is the result rather than the procedure of producing the result. The decomposition from task to its sub-tasks is based on natural language standard, which means each sub-task is still expressed in natural language and is human-understandable. In this procedure, the interaction between machines and service requestors is a critical part in fulfilling the whole task. More specified information which is not represented by service requestor can be fetched during this procedure. For example, it a person would like to make a phone call to BT to subscribe some services. The person's answer to the operator's questions is a way to provide more precise description of what is needed. This decomposition procedure has already been done by the telephone company and what service requestors need to do is to fill in the information according to the pattern to complete the service description. The decomposition on this level can be carried out automatically rather than manually. Voice call service is a typical representation to replace the operator. Service requestors need to follow the instructions and provide further more information by selecting corresponding buttons on the phone pad.

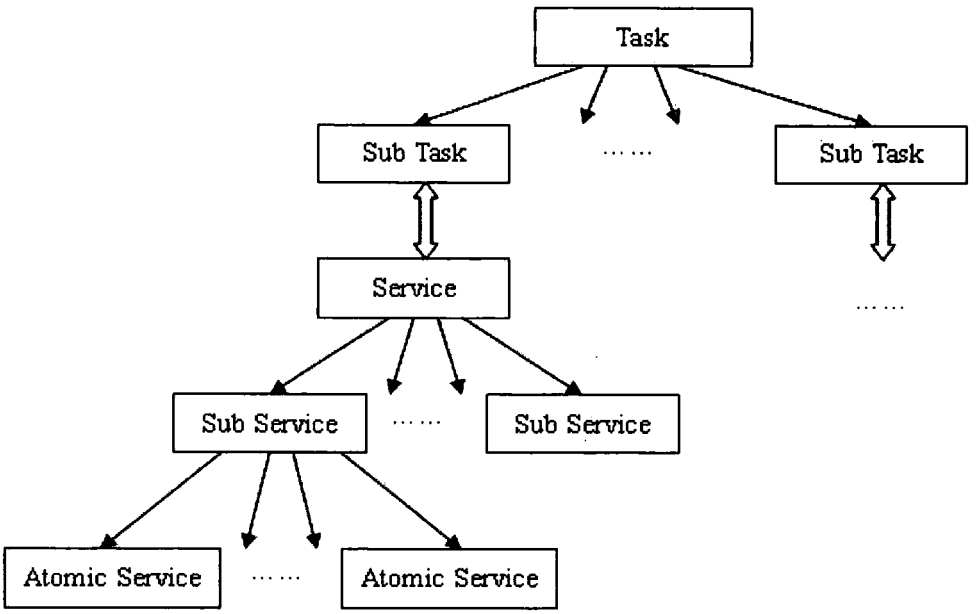


Figure 3-1: Top-down Convention

After several steps of initial decomposition, each task needs to be converted to a functional service whose description is then unknown by service requestors. This, however, is mostly done manually by providers as it is quite hard to find a device which can parse the natural language based description to the machine-understandable pattern. Decomposition below this level aims to fulfil each task by realising the function of each fundamental service and get the final result back to the requestor.

3.3.2 Bottom-up Convention

Unlike service decomposition which is discussed above, the composition of service goes in an opposite way. The composition issue concentrates on fundamental services which can be executed directly and the integration of these services to perform complex tasks. In the perspective of service composition, the method is usually called bottom-up convention.

The mechanism of this kind of bottom-up convention is based on integration principles and flow regulations. Starting from bottom-level service which have single

functions and cannot be decomposed to its sub-services, bottom-up convention follows the service flow which directs the completion sequence of services.

According to the figure, fundamental services are composed together regardless of specific requirements. The purpose of such kind of composition is to generate general service modules which upper levelled requirements can adopt when necessary. However, this is not to say that every two or more services can be coupled. The composition, as mentioned above, needs to follow the regulation of principles such as the output of the current service should be matchable to the input of its follower. The collection of relevant and matchable services forms a service module and the module is restored in a service repository for further use.

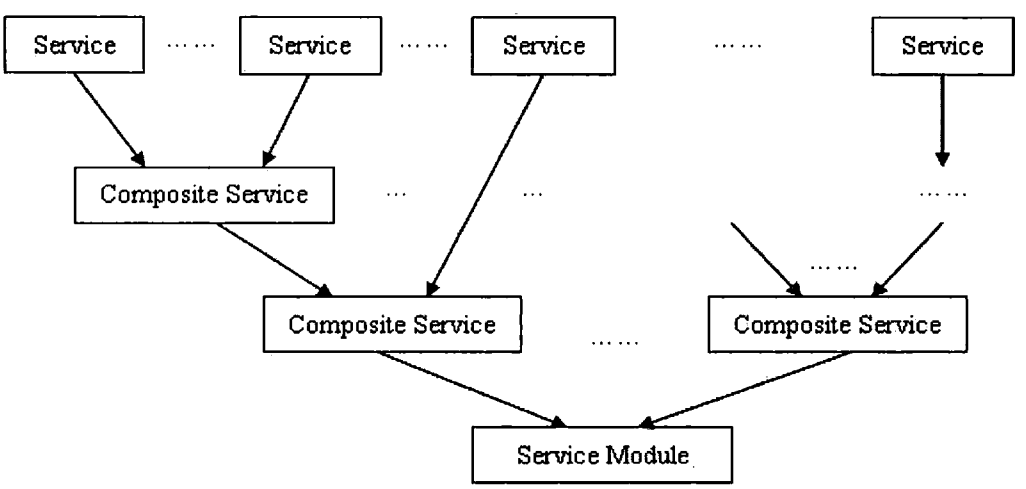


Figure 3-2: Bottom-up Convention

Basic services relating to matchable integration form a composition pair: $<S_1, S_2>$. Several properties of the two services, such as input/output relationship and semantic coherence, need to be satisfied in order to constitute a pair to be composed.

3.3.3 Composition Based on Workflow

Workflow shows the operational procedure of how composed services are situated in the ontology, how they are synchronised and how data is transmitted. It traces the task

process and monitors the execution of service components. There are three aspects for services to be composed to follow according to workflow theory:

(1) The description of input;
(2) Composition rules and algorithms;
(3) The description of output.

Two services can only be composed if the output of the previous service can match the input of the following service. The information of data and semantics is specified in the description of input and out. In another word, not only the quantity and type of data, but also the meaning of services, should be matched in order to be composed. In case that appropriate targeted service does not exist or is not available, composition rules and algorithms are applied in order to search for an alternative service.

3.3.4 Graph-related Service Composition

3.3.4.1 Relationship between Service and Graph

In order to represent our service structure, we use a service graph to illustrate the atomic services and the relationship among them. This is shown in Figure 3-3.

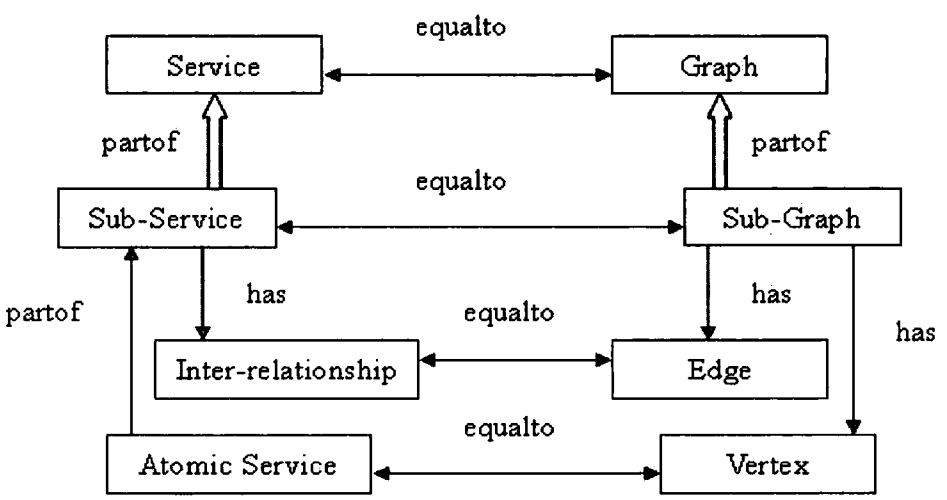


Figure 3-3: Relationship between Service and Graph

A graph is a set of nodes (vertices) connected with each other by some links (edges). If a graph is used to represent a service, a service graph will then be generated. The reason for doing this a service is a composite of a set of sub-services (small services which are decomposed from large ones and can be integrated according to some regulations). Similarly, a graph is made up of a set of vertices and edges, and can be regarded as a set of sub-graphs (small graphs of some meaning which serve as components of bigger graphs). In this way, a correspondence can be defined between a service and a sub-graph. Each node within a sub-graph represents an atomic service, and the relationship between atomic services is represented by edges between vertices. A service which consists of several atomic services corresponds to a sub-graph, and both of them are subsets of a bigger service and graph respectively.

3.3.4.2 A Flight-Booking Example on Decomposition

In order to give a better understanding of the procedure of service decomposition, let us give an example of the flight booking system, which a lot of people may use in daily life.

Suppose a simple booking system is composed of four parts: listing available flights, booking a flight, information validation and check-out. In this case, it means that the service of ticket-booking can be divided into four sub-services with interrelations linked among them. The results of listing available flights are used for booking and the objects of validation are the information provided by service requestors. These four sub-services, therefore, are operated in sequence. In terms of the decomposition from a sub-service to atomic services, let us take listing available flights as an example. It can be divided into several parts as follows: items input by service users (including origination and destination, departing and returning time, etc.), matching information in database and results listing.

By using the technique of graph representation, the booking service and the

relationship between its components can be expressed as in Figure 3-4:

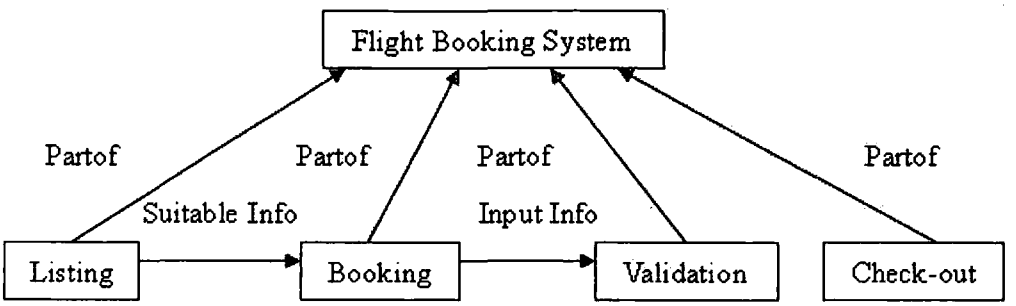


Figure 3-4: Top Ontology of Flight Booking System

Meanwhile, the sub-service of listing available flights can also be decomposed into atomic services as illustrated in Figure 3-5:

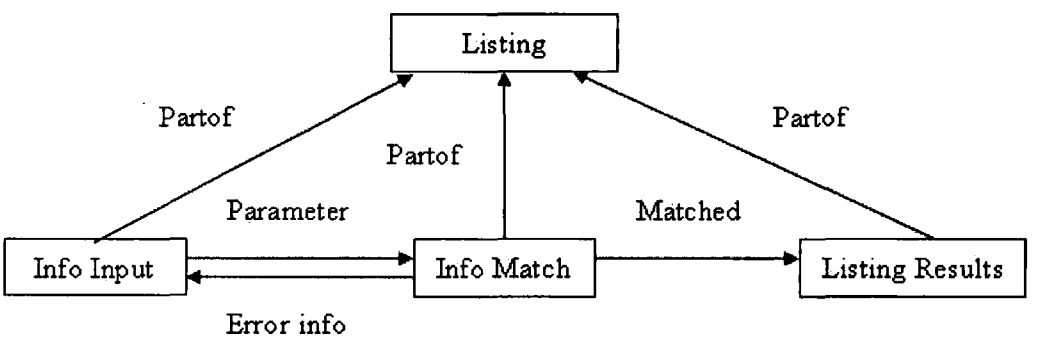


Figure 3-5: Structure of Listing

3.4 Atomic Service

The definition of Atomic Service is presented as a simple and undecomposable service with a lifetime of a single request.

From this definition, it is not difficult to notice at least three meanings:

(1) An atomic service is a service which can perform a function or realise a method, just like all other services;

(2) It can meet a requirement which is uniquely corresponded to;

(3) It is not decomposable, which means it does not contain any sub-services.

A complex service request can only be realised after it is analysed and decomposed to its components and sub-services. Decomposition, however, is not regarded to be as thorough as possible. On the contrary, we assume that the process of decomposition comes up to the end when the object service that cannot be decomposed any further and it should be executed directly. In this way, we put forward the concept of atomic services, each of which acts as a fundamental component of complex services.

According to the definition above, the concept of atomic service is proposed with the aim of the realisation of a single request. Normally, a service request raised by users is not simple enough that a basic function in a programming language can be invoked to fulfil it. After it is decomposed into its sub-services, and deeply to atomic services, the requested service can now be viewed as a composition of more than one fundamental modules integrated together according to integration rules. These modules can be applicable classes or methods in the function lib file. Therefore, these modules, also named atomic services, form a repository of available services which can be invoked by services on upper levels.

3.4.1 Service Type

An atomic service must be of some type. Types of a service can be either data type or object type. Data type refers to the forms of values in terms of input and out, because each atomic service, which finishes in a certain period of runtime in normal cases, has some inputs and outputs to interrelate with other services. Object type means different operations, sharing a same name, in various circumstances. For example, addition is the name of an operation. Here, in other words, it is a kind of atomic service which has certain function. Every addition has at least two operands, which act as parameters or data for this operation. The types of data are the types of addition. Therefore, the data type of the atomic service "addition" includes the type of integer, real and other possible types. For an addition under particular circumstances, such as the addition of two integers, we define its object type as addition of integer. Similarly, other object

types under various occasions can be defined.

3.4.2 Properties

Atomic service, denoted to be AS, has several properties as follows:

(A) Execution Time: t

Each atomic service has its own execution time for completion. Execution time of some atomic services is stipulated in some specifications, such as CPU time. Execution time of other atomic services can be estimated before operation. Execution time of composite services, which are composed from atomic service according to some integration rules, is the aggregation of the execution time of its sub-services.

(B) Input & Output

An atomic service is defined as a service which has one or more inputs but only one output, which guarantees that it will definitely return a result by taking in several values. By representing input as *I* and output as *O*, this rule can be formulated as follows:

$$I(AS) = \{ i_1, i_2, \ldots, i_n \}, n \geq 1$$

$$O(AS) = o_n, n \geq 1$$

From the formulae above, it can be seen that the input of an atomic service is a collection of at least one of all the inputs of the whole service, while the only output of an atomic service denotes to one single parameter which is generated to be either a temporary value or a final result.

(C) Father Node

Since we regard atomic service as undecomposable, it is a leaf-node in the service decomposition tree (a hierarchical tree illustrating decomposition). Furthermore, as an atomic service is the ultimate result of decomposition from complex services, it certainly has ancestors (services which atomic service is decomposed from) which can

be represented as non-leaf nodes. Using f(x) to denote the father node of x, it can be defined as follows:

$$f: f(AS) = \{ S_i : S_i \text{ are non-leaf services } \}$$

Moreover, since every atomic service is the outcome of decomposition, it should have at least one ancestor. In this case, the father node set of an atomic service can never be empty.

$$f(AS) \neq \varnothing$$

(D) Homo-ancestors

Atomic service comes into being because of decomposition. Given an atomic service, one or more than one ancestors can be found in the service decomposition tree. Using AS_1 and AS_2 to denote two different atomic services, with $AS_1 \neq AS_2$, we will discuss the relationship between $f^{\Sigma}(AS_1)$ and $f^{\Sigma'}(AS_2)$. Here, we use the symbol $f^{\Sigma}(x)$ to denote the operation of tracing the father node of x in terms of the decomposition tree repeatedly.

(1) If these two atomic services can be obtained from a topmost service, then we have $f^{\Sigma}(AS_1) = f^{\Sigma'}(AS_2)$, which means the ontology tree of both services is unique.

(2) If not, we have $f^{\Sigma}(AS_1) \neq f^{\Sigma'}(AS_2)$, which means that the ontology trees of these two services are different and they will form a forest to illustrate a task.

(E) Service Pool: Sp

Service pool is a repository which is made up of atomic services and can be represented as follows:

$$\forall a \in S_p : \text{a is AS}$$

Therefore, for every atomic service, it must be a member of the service pool set.

(F) Integration Rules

The process of integrating atomic services AS_1 and AS_2, denoted as $AS_1 \cdot AS_2$, which can perform more complex functionalities.

Integration match-up regulations:

Amount: $O(AS_1) = I_i(AS_2)$, which means that the output of the first service should be one of the input of the second

Type: $T(O(AS_1)) \subseteq T(I_i(AS_2))$, which means that the type of the output must be the same as or a sub-type of the input

3.4.3 Structure

The model in Figure 3-6 describes the architecture of an atomic service, which is the basic component of complex services on the upper level. From Figure 3-6 we can see that there are four fundamental properties of the atomic service: process, object, parameter and agent. The relationship between atomic service and each of these properties is "*has*" and all of these properties serve as exclusive identifiers for a particular service.

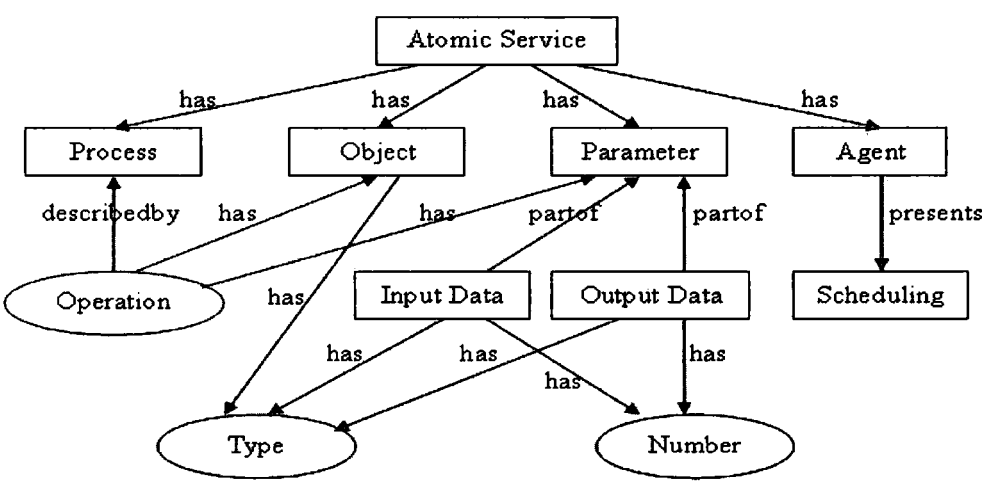


Figure 3-6: Structure of Atomic Service

Process: It is a specification indicating how this atomic service can interact with other services. Being a part of an operation, it illuminates the mechanism of the operation in details.

Object: It serves as an instance of the operation and varies in different circumstances even under operations sharing identical names. Therefore, each object of an operation has its own type to be differentiated from others.

Parameter: Each and every atomic service carries some parameters in the information which it uses to communicate with other services. Data information, in most cases, takes up the majority of parameters. There exist some data input to and output from a service and these data are of some types which are pre-defined by the necessity and environment of the service. The number of data should be taken into consideration on the integration issue of services.

Agent: The agent plays a role of control centre which operates the scheduling issue of all the atomic services. Scheduling, in this respect, means the sequence of running services. Most services use parameters output from previous services as input, and meanwhile, provide results for subsequential services as input. Therefore, the scheduling problem is quite crucial as there will be no proper results if services are arranged in reverse order. For each particular atomic service, the run time is a limited period which is specified in advance. If a service fails to complete within the time period given, the process of exception-handling will be called to solve this problem. Services which are concurrent with this one can be called at this time in order to save time. In this case, the whole service will not collapse because of exceptions caused by one of its components. All of these operations are regulated in the agent.

Besides the description of service structure which is illustrated by Figure 3-6, there is also a linear expression which has the similar function to present the structure. The linear expression is often used in the service catalogue and will be further discussed in

the service scheduling section later.

Let us assume that there is a service of searching for a song on the internet and we can use this service to interpret the model above. The service of finding a song is a composite one and can be decomposed into several atomic services. The way of searching can mainly be based on the name of the song or the name of the singer, which can be regarded as different types of the object of search. The procedure of searching is approximately divided into three steps. First, users type in relevant information they would like to search for. Then, the system takes it as the input data and does the match-up work with its own database. Finally, the result will be output and displayed to service requestors.

3.4.4 Composite service

Composite services are the structured and semantic collection of atomic services. "Structured" means that the collection of atomic services needs to be based on certain regulations and "semantic" means that the collection should be meaning-trackable.

Composite services can have several levels as in most cases, it is impossible that the general requirement can be converted to a direct one-step collection of atomic services. Composite services may have to combine with other composite services, or even atomic services, to make the composition gradually levelled up in order to reflect the requirement with a matchable composite service finally.

However, it is not to say that atomic services cannot be further decomposed indeed. In terms of the decomposition in our research field, an assumption is made that the service decomposition terminates when we reach the atomic level. This is because an atomic service, compared to the original service request, is simple enough in both structure and content to be implemented. Moreover, there is no need to further decompose atomic services to several more simple services or functions. For instance,

in the example above, we specify the bubble sort as an atomic service. Although sorting methods can be normally subdivided into actions such as comparison and swap, there is no need to do so at this stage. Being a basic method in the function library, the bubble sort itself can realise the simple service of sorting by taking input in and giving results, no matter whether the input is in the form of an array or not. According to the definition and properties of atomic service, bubble sort is a service which makes a single response "bubble sort", the name of the service. Therefore, it is regarded as an atomic service within our research domain.

3.4.5 Case Analysis

Here we will display an example in details to further illustrate the concept of atomic service and composite service. The scenario is set up as follows:

Suppose a customer is going to buy bread via an online shop. First of all, he starts searching for the type of bread he wants, putting the exact name of the bread or just a general description in the search bar and expecting the results. The system will return both the exact and relevant items to the customer, who will then begin to either get the list into a sorted order or just pick the item he wants. He can not only get to know the detail of this product, but can make comparison with other products, in terms of the attributes of the product such as price, date of production and time period of reservation. After deciding which one to buy, the customer can put the item into a cart, the place where he can modify the quantity and his purchase history. He can then return to search what else he would like to buy. The customer needs to check out – making payment in some way, which marks the end of the whole shopping procedure.

From the description above, it is not difficult to draw all of the steps of the shopping procedure:

(1). Search for bread

(2). List of all the bread which meets the condition

(3). Sort items in specified order

(4). Pick up the desired type of bread

(5). Put the bread into the cart

(6). Continue to search

(7). Check out

All of the above are components of the whole shopping process, while they themselves are composite service. This means each of them consists of several atomic or composite services and their performance relies on the realisation of those atomic services. Figure 3-7 illustrates the workflow as well as the components of each composite service.

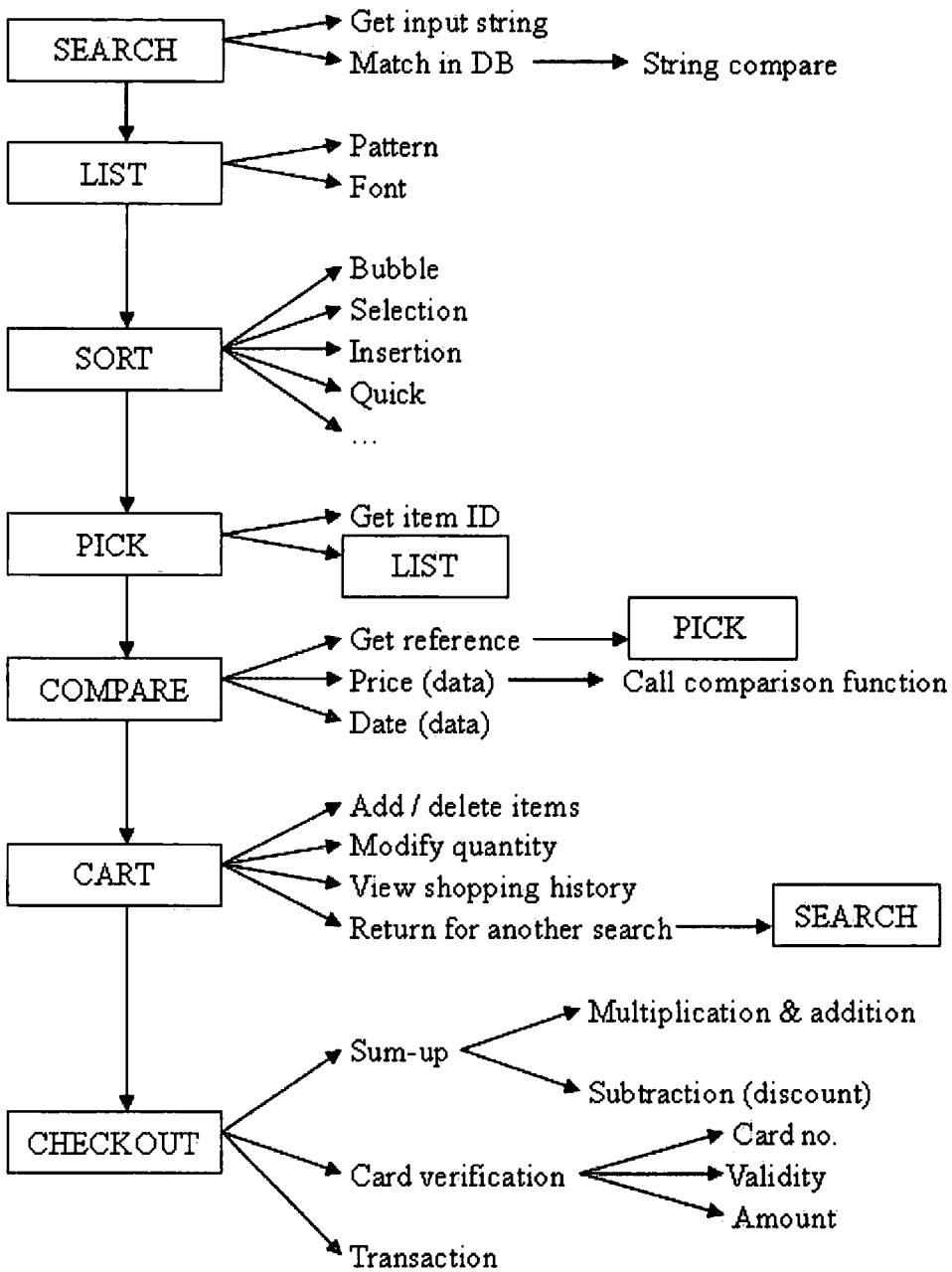


Figure 3-7: Process of Online Purchase

From Figure 3-7, we can see the components of each composite service in details. The components of a composite service can either be an atomic service or a composite service which can be further decomposed. Take the "pick" process as an example. It consists of two components at the first level, "get item ID" and "list". The first component can be regarded as an atomic service as it can be realised by sending a "get" signal to the server and return with the required parameter. The latter, however,

is still a composite service as described in the second step. The realisation of "list" depends on both the display pattern and font, along with parameters. Therefore, it is such kind of composition step by step at each level that finally contributes to the implementation of the whole process.

Meanwhile, it can be inferred from the whole structure that the parameters between processes transmit as follows:

(1) SEARCH: [input string] → get string → string match:

Case 1: true → output item (ID)

Case 2: false → search → output similar item

(2) LIST: [item ID] → LIST → layout (pattern & font)

(3) SORT: price (data) & item ID → SORT → LIST

(4) PICK: item ID → PICK → LIST

(5) COMPARE: [price 1, item 1] [price 2, item 2] → COMPARE → LIST

3.5 Service Description and Matchup Based on Atomic Service

3.5.1 Document-based Service Description

Being the description language recommended by W3C, Web Services Description Language (WSDL) offers description on how to communicate using Web Services. However, it provides no information over the process of liking up one service with another. As a result, an operation can be unaware of being invoked at a certain state to consume Web services. On the other hand, a service can only be identified by its description, which includes service name, parameters and operation etc. These information needs to be "read" by the caller in search before it is invoked. Therefore, we try to save all the necessary of a service in its corresponding document as a tag and the document can thus be checked in interactions. Figure 3-8 gives a brief infrastructure of a document-based service.

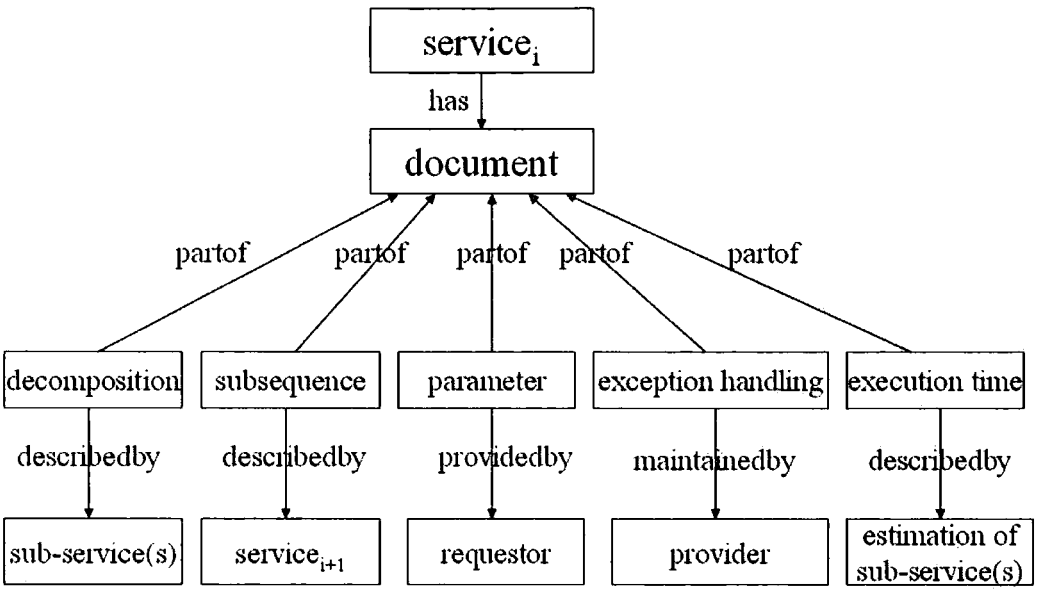


Figure 3-8: Infrastructure of Document-Based Service

According to this structure, each service is allocated a corresponding document. The document has five properties, each of which is "*partof*" the document.

(1) ***Decomposition:*** describes the descendants (sub-services) of the current service and the way of decomposition into them. In case of an atomic service, this property remains null.

(2) ***Subsequence:*** describes the instant following service of this one. The proposition of this property aims to resolve the major deficiency of WSDL discussed above. If the current service is the last component of a composite service, its subsequence will then direct the starting component of the following composite service.

(3) ***Parameter:*** is used to pass information, including both data and non-data type, among services. According to the requirement from the service consumer, the type and value of parameter help find appropriate services.

(4) ***Exception handling:*** This backup process is invoked in case of any exceptions.

(5) ***Execution time:*** Each atomic service has a certain period of time according to its specification. The total time of completing this service is not the simple sum of its sub-services' runtime because of the existence of exceptions, especially the halting

problem. However, there is a certain period of time for each service to be completed, and if the overtime is much longer than expected, it is regarded as the occurrence of exceptions and the handling program needs to start.

3.5.2 Matchup Regulation and Semantic Integration

Now that dozens of atomic services are available, we come to the issue of how to integrate them together into a big service to meet the needs of consumers. Every atomic service has an interface to communicate with each other mainly by transmitting data. The process of integration is not just putting several pieces of atomic services together and adding some relationship between them in order to realise particular functions. The regulations of integration are restricted by the types of input and output.

Each atomic service has its own pre-defined number of input and output data, as well as the data type. The input and output data types of each atomic service k should be an element of the data set of the whole service:

$$I_k \in \mathbf{I},\ O_k \in \mathbf{O}$$

This means that data types of each atomic service can not be types that are not defined at the beginning.

In order to integrate several pieces of atomic services, the number of input and output data should exactly match. For a particular atomic service, the number of data output from previous services must be equal to the number of its input, assuming that all of the data from previous services be transmitted only to this service. And the post-condition of the integration is that the number of data output from this atomic service and its peer should match the number of data of their following services. This mechanism can guarantee the correctness of data flow within the whole service and produce final results on the input data given as initial values.

The data transmission through the combination of services normally follows one-way data exchange pattern. This is to say that the previous message which sends the data does not expect any response from its following service which receives the data. Therefore, the data transmission is not guaranteed and unreliable as the sender has no idea of whether the data has successfully arrived at the receiver. However, the system becomes less vulnerable when applying a request/response message exchange pattern during service scheduling.

On the other hand, the expected result may still not be met even if two services can be composed together after they satisfy the number restrictions on parameters. In order to let the integration produce consistent result, the descriptions of those two services to be composed need to be taken into consideration as well. Because the decomposition of the service requirement has been done at the service analyse stage, the appropriate services will thus be invoked in the sequence according to the decomposition document. Each atomic service has its own description, apart from necessary properties, which gives the information on the function of the service. Therefore, when it comes to the stage of selecting services with appropriate functionalities, the description of the service needs to be pattern matched with the requirement, for example by using keyword matching. Based on this, the service invoked becomes a component of the composition at a higher level and the whole service module.

3.6 Scheduling

When a service user raises a requirement, they not only would like to get back the expected result, but also in many occasions, the duration between the request and the feedback should be taken into consideration to some extent. The accurate length of service execution time, however, is often hard to tell because some unforeseen errors may occur during the whole process. Nevertheless, in the point of view of a service producer, a certain time scheduling mechanism needs to be examined to manipulate and monitor the whole process of executing services. In this section, a service

scheduling model and scheduling language, based on the above-mentioned service structure and description, will be introduced in order for selecting and arranging required services.

3.6.1 Scheduling Model and WSSL

3.6.1.1 Scheduling Model based on OWL-S

Based on the structure of service description illustrated in previous sections, we put forward a model which is aimed to be applied in scheduling.

The linear structure of a service, which has the similar function as the graphical expression, is more efficient in terms of enumerating services which are about to be scheduled. It can be represented as:
S [name, parameters, description, operations]

The name of the service is a nominal symbol for recognising the identity of service while the last three characteristics all contribute to the scheduling structure.

Parameter [input, output, type, precondition, effect]
Description [time, subsequence, precondition, decomposition, resource]
Operation [trigger condition, exception handling, lifetime]

First of all, we use OWL-S to describe the service ontology.

```
<!-- Service -->
<owl:Class rdf:ID="Service">
  <rdfs:label>Service</rdfs:label>
  <rdfs:comment>General Service Structure</rdfs:comment>
</owl:Class>
```

```
<!-- Parameter -->
<owl:ObjectProperty rdf:ID="hasParameter">
   <rdfs:domain rdf:resource="#Service"/>
   <rdfs:range rdf:resource="#Parameter"/>
</owl:ObjectProperty>

<!-- Description -->
<owl:ObjectProperty rdf:ID="hasDescription">
   <rdfs:domain rdf:resource="#Service"/>
   <rdfs:range rdf:resource="#Description"/>
</owl:ObjectProperty>

<!-- Operation -->
<owl:ObjectProperty rdf:ID="hasOperation">
   <rdfs:domain rdf:resource="#Service"/>
   <rdfs:range rdf:resource="#Operation"/>
</owl:ObjectProperty>
```

The above segment is the top ontology of service. We can further describe the each property of service according to OWL-S. For example, in the proper of parameter, the input, output and their types can be defined as follows:

```
<!-- Input -->
<owl:ObjectProperty rdf:ID="hasInput">
   <rdfs:subPropertyOf rdf:resource="#hasParameter"/>
   <rdfs:range rdf:resource="#Input"/>
</owl:ObjectProperty>

<!-- Input Type -->
```

```
<owl:ObjectProperty rdf:ID="hasInputtype">
   <rdfs:subPropertyOf rdf:resource="#hasInput"/>
   <rdfs:range rdf:resource="#Inputtype"/>
</owl:ObjectProperty>

<!-- Output -->
<owl:ObjectProperty rdf:ID="hasOutput">
   <rdfs:subPropertyOf rdf:resource="#hasParameter"/>
   <rdfs:range rdf:resource="#Output"/>
</owl:ObjectProperty>

<!-- Output Type -->
<owl:ObjectProperty rdf:ID="hasOnputtype">
   <rdfs:subPropertyOf rdf:resource="#hasOput"/>
   <rdfs:range rdf:resource="#Outputtype"/>
</owl:ObjectProperty>
...
```

3.6.1.2 Web Services Scheduling Language

Having defined all of the properties, contents and relationship, the Web Services Scheduling Language (WSSL) is proposed in order to describe the scheduling procedure.

Based on service description language and flow language, we design the WSSL to specify services and the scheduling process, which are illustrated as follows:

(1) **Describe WSSL:**

```
<!-- Definition -->
<wssl:definitions name="Service Scheduling" namespace="URI">

<!-- Description -->
<wssl:description>
  <wssl:document ... />
<wssl:entity>
<wssl:admin name="Admin">
<wssl:service name="Service Name">
<wssl:timer name="Timer">
  </wssl:entity>
  <wssl:signal>
<wssl:start name="Start">
<wssl:end name="End">
<wssl:mutex="True"|"False"|null>
</wssl:signal>
</wssl:description>
```

The general description expresses the fundamental scheduling properties of each service, which specifies both the service manipulator (admin) and the service itself. It also initialises a timer and a signal which are used for the communication between admin and service so that the admin is able to monitor the whole scheduling process.

(2) **Define operations:**

```
<wssl:operation>
  <wssl:exchange name="Name" source="Source" target="Target">
  <wssl:send name="Name" value=Value type=Type>
  <wssl:receive name="Name" value=Value type=Type>
</wssl:operation>
```

This is the standard definition of service operations. It includes the name, origin and destination of a service command, and specifies the format of both the input and output, represented by send signal and receive signal respectively.

(3) **To start invoking a service:**

```
<!-- Start Service Invocation -->
<wssl:exchange name="Start" source="Admin" target="Service_i">
   <wssl:send name=start value="Start" type=signal>
   <wssl:receive name=mutex value="True" type=signal>

<!-- Start Timer -->
<wssl:exchange name="Start" source="Admin" target="Timer">
   <wssl:send name=start value="Start" type=signal>
   <wssl:receive name="Time" value=CurrentTime type=time>
```

In order to call a service, a start signal needs to be sent to the corresponding service from the Admin. The variable named "mutex" is an analogue signal which uses logical values True or False to indicate whether the required service has been invoked or not. Meanwhile, the time controlling mechanism is intrigued by starting a timer to record the lifetime of the ongoing service.

(4) **Service Completion:**

```
<!-- Normal Termination -->
<wssl:exchange name="End" source="Service_i" target="Admin">
   <wssl:send name=end value="End" type=signal>
   <wssl:receive/>

<!-- Stop Timer -->
```

```
<wssl:exchange name="End" source="Admin" target="Timer">
  <send name=end value="End" type=signal>
  <receive name="Time" value=null type=time>

<!-- Invoke Following Service -->
<wssl:exchange name="Start" source="Admin" target="Service_{i+1}">
  <wssl:send name=start value="Start" type=signal>
  <wssl:receive name=mutex value="True" type=signal>
```

When the operation of the service is over, an end signal is sent from the service to the Admin. The Admin, accordingly, needs to stop the timer which is used to monitor the previous service on receiving the end signal. Moreover, it needs to prepare the consequent procedure by sending a start signal to the following service in order to instruct the operation.

(5) **Exception Handling**

```
<!-- Error in End Signal -->
<wssl:exchange name="End" source="Admin" target="Timer">
  <send name=end value="End" type=signal>
  <receive name="Time" value=0.00 type=double>
<wssl:exchange name="End" source="Admin" target="Service_i">
  <wssl:send name=end value="End" type=signal>
  <wssl:receive name=mutex value="False"||null type=signal>

<!-- Intrigue Exception Handling System (EHS) -->
<wssl:exchange name="Start" source="Admin" target="EHS">
  <send name=start value="Start" type=signal>
  <receive name=mutex value="True" type=signal>
```

```
<!-- Invoke Alternative Service -->
<wssl:exchange name="Start" source="Admin" target="Service_i'">
  <wssl:send name=start value="Start" type=signal>
  <wssl:receive name=mutex value="True" type=signal>
```

Providing each step of the whole process runs smoothly, the entire service call is able to complete within the expected time limit. However, if a malfunction occurs or the first-choice service is not available, a corresponding exception handling system needs to be invoked by the Admin. If no end signal of a service can be produced within its lifetime, the Admin reckons that an exception occurs. Therefore, it needs to invoke the corresponding exception handling system of this service. Meanwhile, the Admin is also obligate to call an alternative service according to the preference list of services which is generated during the service analysis period.

The WSSL handles the execution of service scheduling mainly according to the five segments described above. It sets up a standard way to organise services and is able to monitor the whole scheduling process.

3.6.1.3 The Significance of WSSL in Service Scheduling

The proposed Web Services Scheduling Language aims to build up a set of rules specifying the communication between services which need to be invoked for composition and the admin. The communication is designed to be based on signal transmission, which carries information from both the sender and the receiver. The WSSL has, therefore, at least three major features as follows:

1. Embed the linear structure of service description into service scheduling process;
2. Indicate the dynamic procedure of service scheduling in terms of service invocation and monitoring;
3. Apply the exception handling session into the scheduling language in order to process the composition under abnormal circumstances.

The WSSL proposed by us sets up standard in terms of operations on the services which need to be scheduled and invoked. It links the dynamic part of scheduling – service communication with static descriptions and facilitates the interoperations between services and admin, the issue which has yet been discussed in relevant researches.

3.6.2 Fashions of Scheduling

Each atomic service can only respond to a single request and realise one simple function. These fundamental services need to be integrated in order to fulfil complex tasks. Different ways of allocating services provide different functions and the styles of scheduling services play an important role in service composition and flow control.

3.6.2.1 Sequential

Services which operate one after another are called sequential services as illustrated in Figure 3-9.

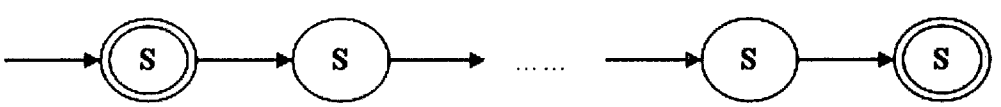


Figure 3-9: Sequential Services

This fashion indicates the chronological order of services in terms of the time sequence of being operated. In this case, each service can only be "active" when its predecessor finishes operation. According to scheduling session, previous services inform the admin and the admin invokes the following service. Each and every service is implemented as on an assembly line.

3.6.2.2 Parallel

Some services, however, need to be implemented concurrently rather than

sequentially in order to perform certain tasks as illustrated in Figure 3-10.

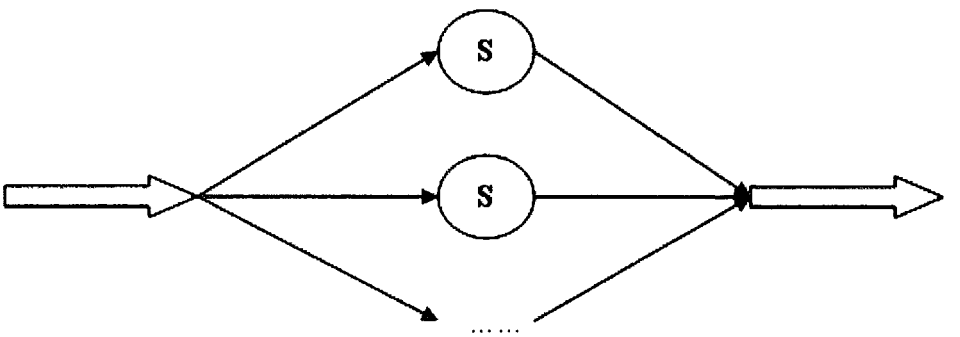
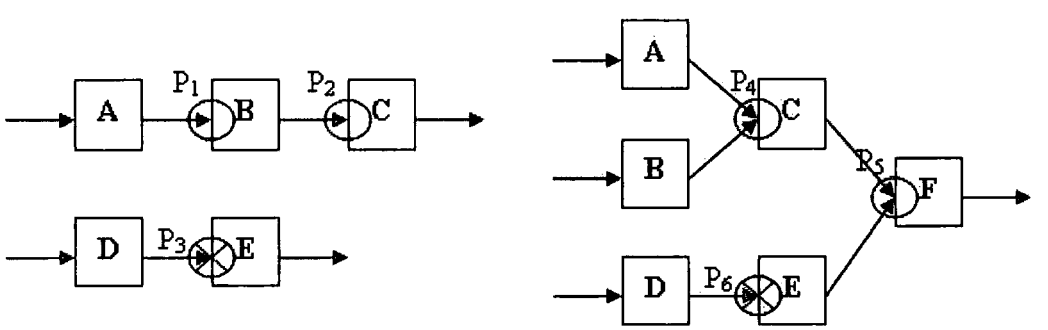


Figure 3-10: Parallel Services

3.6.2.3 Critical Points

Services are scheduled together to be invoked in certain order, either sequentially or parallelly and possibly, two services run individually with no relation. The invocation of some later-scheduled service may rely on the result of previous ones. Therefore, this kind of service has to wait the completion of its predecessors. This dependent relationship between services is referred as critical point in service scheduling and is illustrated in Figure 3-11.



(a) Sequential Services **(b) Parallel Services**

Figure 3-11: Critical Points

As sequential services described in figure 3-11(a), service B is the direct follower of A

and the invocation of B totally relies on the result provided by A, thus P_1 is a critical point to A. Service C, which does not follow A directly, depends on the completion of B which is the direct following service of A. Therefore, service C indirectly depends on A, thus P_2 is also a critical point. However, the implementation of service E has nothing to do with A. Although E is scheduled after A, P_3 is not a critical point.

Now let us turn to parallel services which are displayed in (b). Because of different operation time for each service, it is possible that the finish times may vary. In the case that the following service needs the overall output from these concurrent services, the admin needs to wait until the completion of the most time-consuming service and invoke the following service. For example, the results from both parallel services A and B contribute to C. Therefore, C has to wait both of them to finish so as to be invoked. The waiting time is the operation of either A or B, depending on which one costs more time and obviously, is a P_3 critical point to A, and so is P_5. Meanwhile, if a later-scheduled service does not need the previous output, it can be invoked ahead of the completion of those parallel services. For instance, although the result of service E will be merged with the output of C, indirectly from A, to intrigue service F, the invocation of E does not depend on A. Therefore, P_6 is not a critical point.

3.7 Summary

This chapter has made a deep research in Web Services mainly in three aspects: service structure and description, service composition and scheduling. We proposed the concept of atomic service and redefined the structure of Web Services based on atomic service. The purpose of the redefinition is to facilitate composition and scheduling. After that, we introduced the two patterns of service composition and illustrated the idea by giving several examples. We also set up a relationship between graph and service, using conceptual graph to represent service and composition. Last but not least, the scheduling mechanism was put forward. With the aid of scheduling model and language, we were able to manage each component in service scheduling.

Chapter 4 System Design and Implementation

Along with the solid evolution of issues such as service description, composition and matchmaking, more and more systems which can carry service properties and are capable of realising business tasks have been developed in recent years, represented by IBM Websphere (BPEL server implementation), OMII (a Web Services infrastructure relating client, server and repository) and myGrid (data-supportive middleware components). Similar to the two divergences of Web Services, service-related systems also consist of variations such as functional end and business end.

4.1 Synoptic Analysis

4.1.1 Functional End

From the perspective of fundamental services composable to perform complex tasks, the outcome of realising atomic services and composite services at functional level is to produce fundamental service modules for business processes to invoke. For example, basic classes in the function library are abstract and may need to be bound with other classes to form a function or method to correspond to an invocation. The aim of composition at functional level is to make machines have the ability of selecting the corresponding atomic service or an appropriate one from several similar services, for example, selecting a better method from a list of sorting algorithms. The procedure of building up from these fundamental services is to integrate distributed functions, from abstract classes to task-indicative services.

4.1.2 Correlations of Atomic Services

As mentioned in Chapter 3, Atomic Service is the fundamental service unit in the service ontology. Its status at the bottom level of the hierarchy means that it serves as the basic component of any operation on services. Through the procedure of being

composed together, the functions of those atomic services can be imposed on the composite service. The topmost composition from the functional end is linked with a business requirement at the business end.

4.1.3 Business End

The realisation of Web Services focuses on connecting software components or applications as middleware, offering search and matchup mechanism to bind appropriate services with requirements, and managing the workflow which structures and synchronises tasks. More and more companies, such as Google and Amazon, have presented interfaces for people to use their services. Service users are abstracted from implementation details through these kinds of projects. However, in order to meet requestors' demands, service providers have to set up a relationship of business-end requirements with those functional-based components to fulfil the task.

4.1.3.1 Service Catalogue

In most cases, service providers present an interface which involves catalogues and brief description of their services. Service consumers, therefore, need to pick up their desired services.

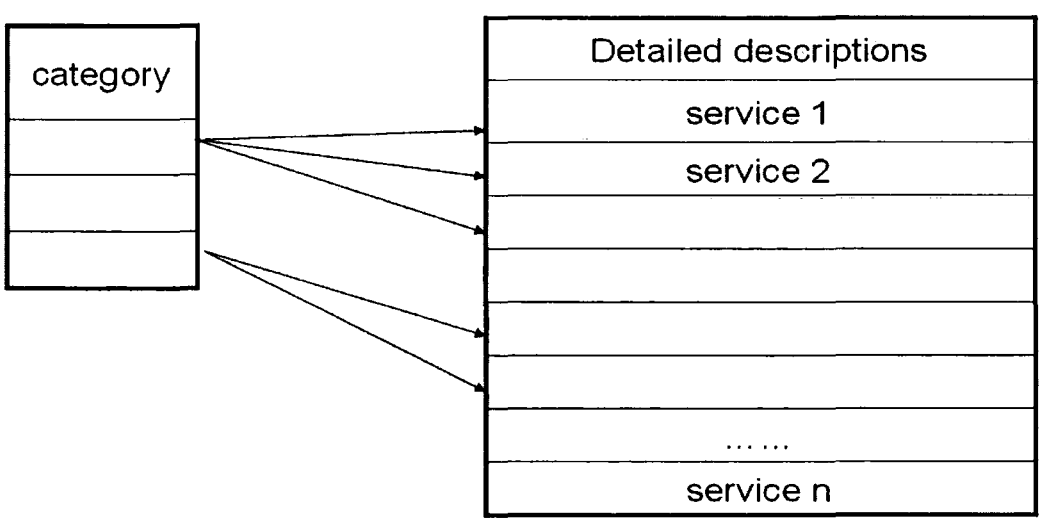


Figure 4-1: Index of Category

Figure 4-1 can be illustrated in two different ways. It can be viewed as a skeleton of a service category which is displayed to service consumers. Services are classified and all of them, together with its sub-services, have detailed descriptions for consumers' reference. Moreover, it is also regarded as a brief depiction of a service repository stored in business registry. The category on this level, however, is opaque to consumers, machine-understandable and is used for searching services.

4.1.3.2 Business Integration

Corresponding to business service requirements and based on functional service components, business integration is dedicated to integrating pieces of enterprise applications and software in business environment. It at least involves data information integration, which ensures that distributed information can be coordinated successfully, and process integration, which links business processes and applications over various platforms.

One of the construct used in business integration is enterprise service bus which provides foundational services for complex architectures. However, it is based on standards and is not used to implement SOA, thus not Web Services-based either.

Among those Web services products developed by various companies, IBM WebSphere is a typical representative for defining middleware software category. It aims to integrate business applications by using open standards such as J2EE, XML, and Web Services. It uses a business integrator to specialise in collaborating software components for implementing business processes both at the enterprise level.

4.1.4 Interrelationship between Both Ends

A service requirement can be met provided the fact that both the functional end and business end of the web service framework work coherently. A requirement at the business end usually comes expressed in natural language, which needs to be

interpreted into machine-recognisable language and thus be implemented at the functional end through the composition of atomic services. This process, therefore, needs the matchup between service at the business end and the one at the functional end, with not only the names of the services should be corresponded, but the functions and other additional clauses, such as input and output, need to be satisfied.

4.2 Dynamic Sessions

The whole process of executing a task can be viewed as a combination of several sessions, during each of which several components of the whole service are dealt with. There are three main sessions in executing services and the architecture of the sessions is presented by Figure 4-2.

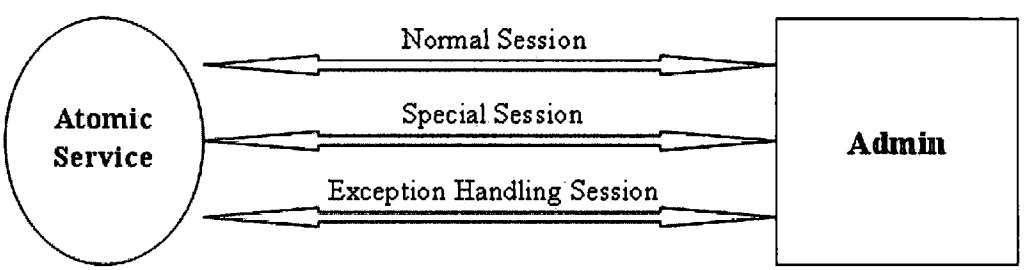


Figure 4-2: Scheduling Sessions

According to the figure above, sessions are made up between admin and services. Under most circumstances, the normal session is invoked, while special session and exception handling session play a part in case of abnormal occasions.

4.2.1 Normal Session

In order to monitor the whole execution of each project, there is a so-called "Admin" to control the process of workflow as a central agent. The Admin plays an important part in initiating services, keeping track of the process of services and also, managing exceptions that may occur within the whole procedure. The process is shown in

Figure 4-3.

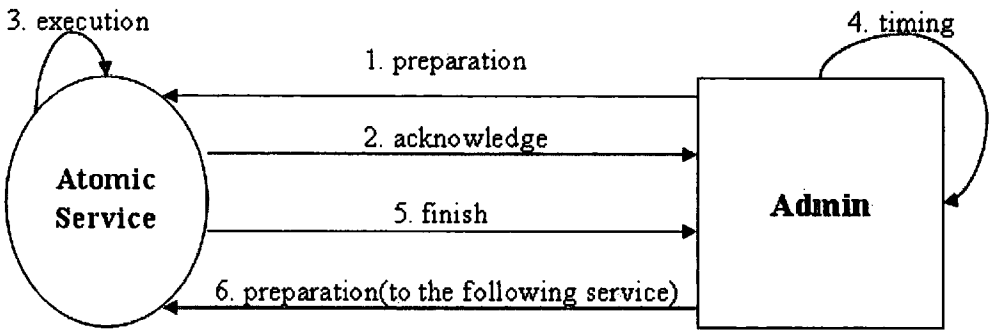


Figure 4-3: Normal Session

When a requirement arrives, it will be firstly reported to Admin. When Admin receives a requirement, it analyses the description of service which has been decomposed and is ready to be invoked. At the very beginning, Admin initiates the first service by sending a piece of message. When a service is invoked, first of all, according to its description, it needs to be examined whether this service can be further decomposed or the service is already a leaf node in the decomposition tree. These two situations should be dealt separately simply because a "leaf-node" service (or atomic service) can be executed by directly calling the corresponding function or method. If the service is not atomic, there is no existing function corresponding to it. In this case, the process of decomposing the service is required because it is a service consisting of its sub-services.

When the expected service is ready to execute, it will send an acknowledge signal back to the admin to report its status. This means that the messages exchanging between the admin and services follows a request/response pattern where the sender forwards a question to the receiver and then waits for the answer to the question. Meanwhile, the service enters the running mode. The admin, however, after receiving the acknowledge signal, needs to start the timing device to keep track of the running service. Each atomic service is allocated a running time and is specified in some

specifications.

When the ongoing service completes operation, it sends a finish signal to the admin. The admin, then, check the service schedule and send the preparation signal to invoke the following service.

4.2.2 Special Session

Each atomic service runs smoothly under the control of the admin according to the normal session. Services are executed either one after another in sequence or at the same time for those parallel services. However, it is not always the case. Even a minor fault could spoil the process of the whole service. Under this circumstance, the special session is activated to deal with these issues.

Because of the congestion or malfunction in the process of transmitting signals, it is possible that either the service cannot receive the preparation signal or the admin cannot receive the acknowledge signal. In this case, the admin needs to re-send a preparation signal to the service after a period of time, which is determined by specific regulations. This mechanism, however, will inevitably increase the load of transmission channel but can inspect if the service has some interior problems when the channel is free. The service does not need to make response to duplicate signal as long as it replies a preparation to the admin.

Therefore, the service will discard the preparation signal it receives in case that it has already sent an acknowledge signal to the admin. Under the circumstance that the service is available to send acknowledge signal and the Admin can receive it, the normal session will apply similarly as stated above.

However, if a service fails to make response to the original signal and the duplicated one through an unoccupied channel, the admin will then make the assumption that the

service is currently not available and prepare to invoke the alternatives of this service. This procedure, as illustrated in Figure 4-4, is the same as the normal session, with the only difference of the object service which the preparation signal is sent to.

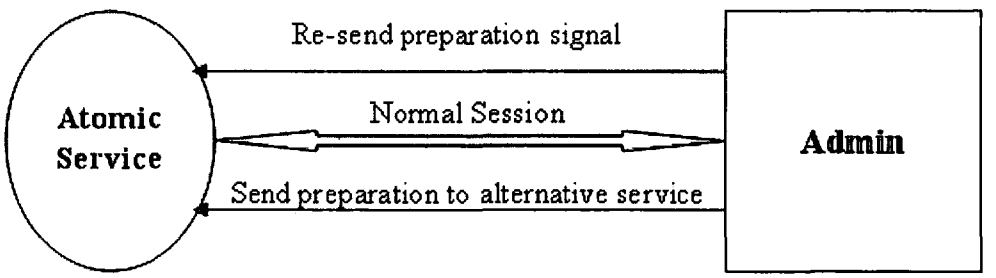


Figure 4-4: Special Session

Let us suppose in the online purchase example bubble sort is initially selected as the sorting algorithm, but it catches an error during the session and can neither complete the calculation nor return a signal to the admin. The admin, therefore, needs to invoke an alternative service to replace bubble sort, for instance, selection sort. In this occasion, the alternative algorithm may work well as a substitute, but the performance may not a factor of thorough consideration, as the best case running time for selection sort is $O(n^2)$, worse than bubble sort's $O(n)$. Subjectively speaking, however, the priority of successfully finding an alternative service is higher than comparing the performance between them. Therefore, it may reduce the damage caused by malfunction of the originally-selected service by invoking a replacement in the special session.

4.2.3 Exception Handling Session

Once the original-targeted service fails to execute correctly, the optimal way is to find an alternative service in the service pool as stated above. But in the occasion of no available replacement of the original service, it is most probable that a real exception occurs. In this case, exception handling session needs to come into operation.

A service could be regarded as a failure mainly in two ways: no response for sending back results or presenting unexpected value. In this section, we will focus on the first occasion and the second one, which associates with correctness check mechanism, will be discussed later.

As mentioned above, the special session deals with re-invoking services or calling alternative services in case of the original ones fail to make response to any commands.

It could be caused by the following reasons:
(a) The collapse of service itself;
(b) Loss of signal in transmission (including invoke signal and acknowledge signal);
(c) The malfunction occurring at the admin side.

The third reason, however, is beyond our research field as the exception at the admin side has little relationship service scheduling issues. Therefore, we only talk about sessions with the admin running regularly as the precondition.

4.3 Visualisable Service System

In order to demonstrate the feasibility and operability of composing from atomic service and illustrate the scheduling mechanism, we have implemented a display system based on hierarchical structure and proposed service description. Its purpose is to present how services are selected, composed and scheduled. Figure 4-5 gives the general interface of the system.

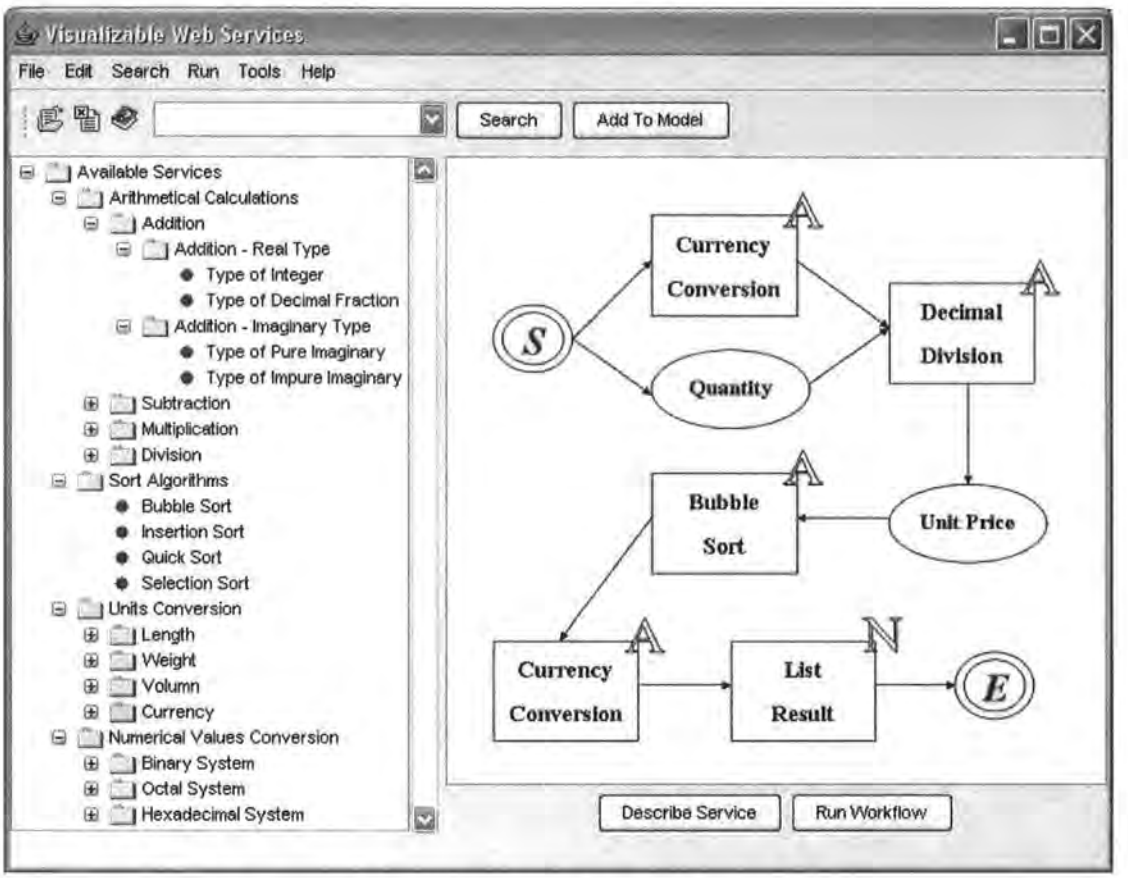


Figure 4-5: GUI of Visualisable Service System

4.3.1 System Introduction

After a general requirement has been decomposed further down to the atomic service level, appropriate services need to be selected to be composed to fulfil the task. Services are originally stored in the service pool, a repository which is categorised according to the basic functions of services. As listed on the left panel, atomic services can be view in a hierarchical order, which is refined to types of input for an atomic service. The search bar above is provided for finding desired services. Once an appropriate service is found, it can be selected to be added to the model, which is presented on the right-hand side. The description of the selected service can be viewed by clicking the button below.

Both the services and objects are intuitively represented in graphics. The service which has been selected from the left panel appears in a rectangle, while all objects

are shown in ovals. The symbols at the corner of services mark the decomposability of each service, as "A" standing for atomic service (for example, bubble sort in Figure 4-5) and "N" for non-atomic service (for example, list results in Figure 4-5). Non-atomic services can be expanded to atomic services according to description. The two circles, S and E, mark the start and end for the whole process.

The example of sorting the unit prices of some products in different currencies is shown above. The procedure starts with converting the currencies to the same one to get the unit price of each one. After that, the array of prices is input to a sort method, for example, bubble sort. As mentioned above, although bubble sort is an algorithm based on swap, but it can be view as atomic service within out research domain. With the sorted prices, a service currency conversion is invoked again to change the currency back to original. At last, the whole process terminates with by listing the final sorted result.

It can be seen that arrows not only denote the direction of the process, but also represent in which way data information is transmitted between services. The completed process can be saved as a service module for further use, as it is not necessary to reconstruct those atomic services again to realise a same task.

4.3.2 Features

At this stage, the system only features the bottom-up convention of service composition. This is to say, atomic services can be composed with ignorance of the service requirement. However, this does not mean that services can be composed arbitrarily. The composition of services needs to follow the matchup and integration rules discussed in Chapter 3. For example, if the input amount of a service is not satisfied or the type of the output from a previous service does not match the type of input of its following service, the system will report a runtime error when operating the whole process.

4.3.3 Illustration of Service Composition and Scheduling

This demonstration gadget is capable of constructing composite service by means of selecting appropriate atomic services in the service pool according to the service requirement. During the process of such composition, the system can handle different types of integration by applying corresponding scheduling algorithms.

4.3.3.1 Sequence Scheduling

Atomic services in order to be invoked and operated in a chronological order form a service queue. In this queue, each service, except the initial one, can only come into action based on result of its ancestor service. The demonstration system is able to illustrate the whole set of atomic services by applying queue scheduling method.

For example, providing there is a service which manages adding a selected item to the shopping cart, the system will schedule the procedure presented by Figure 4-6. In order to implement this service, appropriate atomic services need to be selected from the service pool and then wait to be scheduled. After that, corresponding services can be integrated together according to service description and matchup rules, which have been discussed in the previous chapter. At last, this service requirement is represented as follows: get the product number of the item chosen, find this item in the shopping cart and increase the amount, and list the result. All the services are scheduled sequentially throughout the whole procedure.

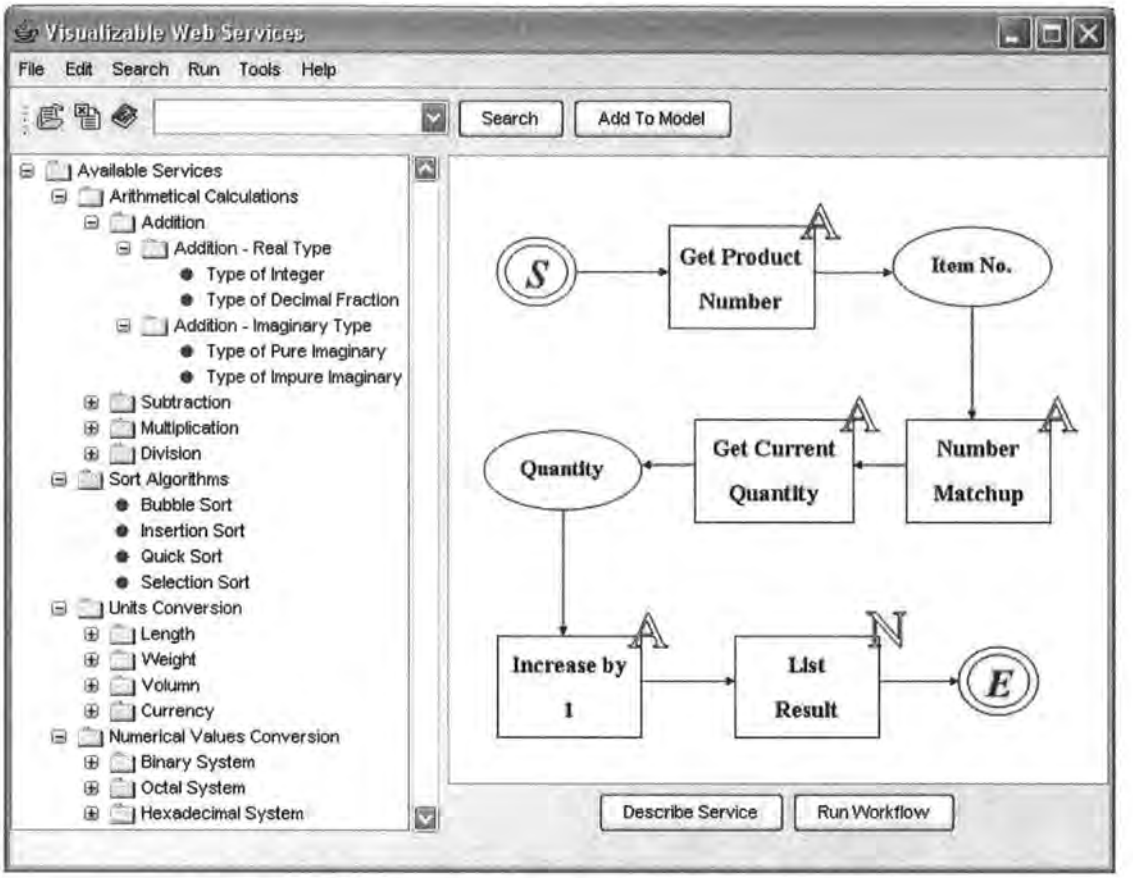


Figure 4-6: An Example of Sequence Scheduling

4.3.3.2 Priority Scheduling

On the other hands, the execution of a service can be prioritised in terms of service flow. For example, a service aims to compare the prices of two items, with the price of item A in sterling while B in another currency. As seen in Figure 4-7, it takes fewer steps to get the unit price of A, as it can be obtained directly from applying the division service by using the item price and weight as inputs. However, the execution of A needs to halt at this point to wait for the unit price of item B (in foreign currency) being changed into sterling. It is because the unit price of B (in sterling) is also one of the inputs of the following service and thus becomes critical point in terms of scheduling fashions. The execution of the atomic service currency conversion, in the shaded rectangle, is prioritised ahead of other services. Therefore, the priority scheduling is able to guarantee the parallel services and the whole service flow.

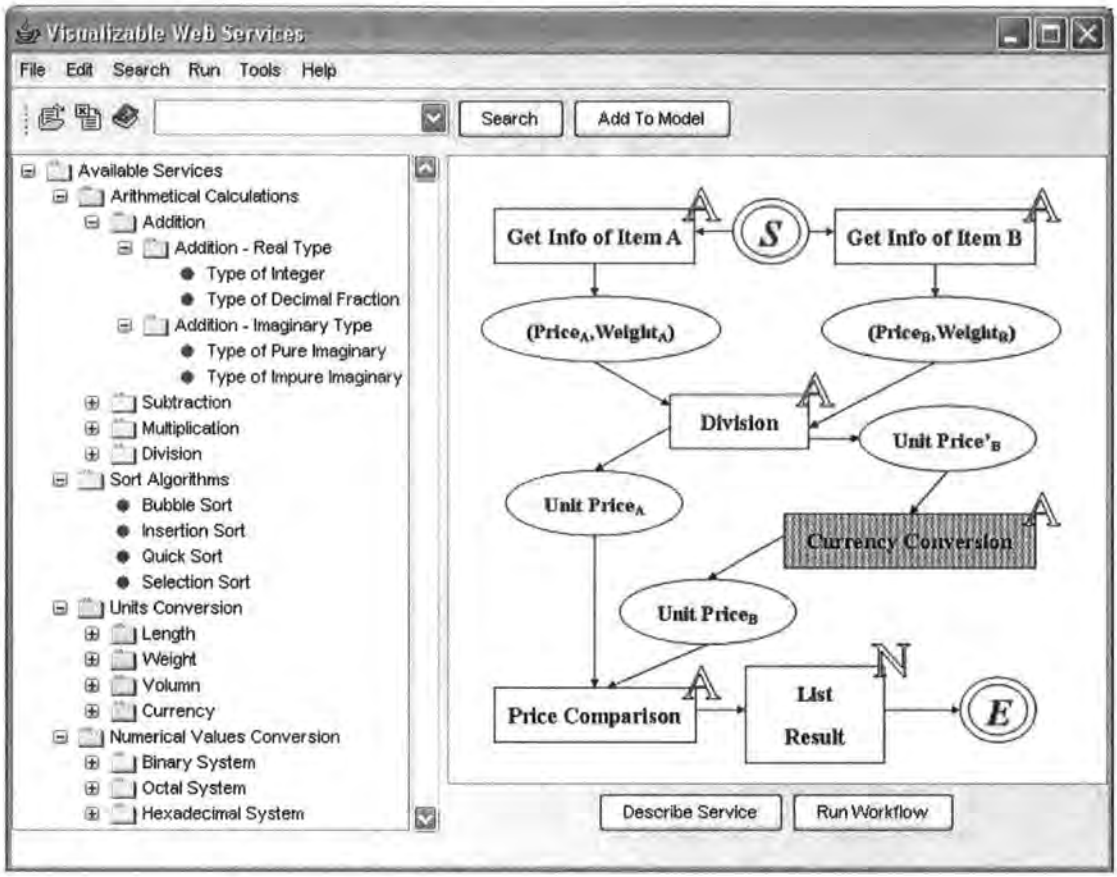


Figure 4-7: An Example of Priority Scheduling

4.3.4 Future Improvement

In future work, the top-down convention of composition issue needs to be added to the function. In another word, given a composite service as a service module, the system needs to decompose it automatically and recognise every decomposed atomic service. The service repository, therefore, needs to be expanded in scale for all time so as to cover as many atomic services as possible and does not reject decomposition if a certain decomposed atomic service does not appear in it. Moreover, the semantics of composition is necessary to be considered. Currently, any services which are able to satisfy the composition rules can be combined together without verified by semantics. As a result, this may generate some meaningless composite services of no use. Therefore, the system should have the ability of applying semantic check in the meantime of approving services to be physically composed.

4.4 Summary

This chapter mainly focuses on the realisation of service composition and scheduling. It first analyses the both ends, functional and business, of Web Services and draws the service catalogue, which is referred for creating service pool. It then presents three main dynamic sessions in the service execution by discussing how services are correlated during composition. Based on the service pool and composition mechanism, a demonstration system is introduced in order to feature the composition and scheduling procedures. By use of scheduling algorithms, the system illustrates how services are composed under different circumstances.

Chapter 5 Reasoning and Evaluation

5.1 Theory Basis

Subjectively speaking, the object of operations like composition and scheduling is service. The operations imposed on services make use of the relationship between them. Even the relationship between business services can be viewed as the integration of multiply relationship among fundamental services. Here we will make a deep research on functional services to find out inter-relationship between them and how they are collaborated.

5.1.1 Evidence: An Example on Composing Basic Functions

Web Services can be divided into components, each of which may have some relationship with others. Moreover, even those basic classes within the function libraries will interact with others when they are invoked. In order to make a clear explanation, we present an example of the public class MouseEvent, which extends InputEvent, to show the relationship between classes. The reason for choosing MouseEvent class is that several mouse event-related classes will be invoked upon a mouse action, which can be viewed as atomic services required in order to perform a complex task in the perspective of service composition and scheduling.

(1) ***Description***

Mouse events are fired by a component when the user interacts with the component using the mouse. For example, when the cursor enters the component bounds, a MOUSE_ENTERED event, which is one of the Mouse Event Ids Constants of MouseEvent, is fired and when the user clicks a mouse button, a MOUSE_PRESSED event is fired.

(2) ***Listening for Mouse Events***

To listen for mouse events from a component, the listener must implement the Mouse-Listener interface. After that, the listen must be registered with the component. It becomes registered by calling the component's *addMouseListener ()* method, one of the Event Listener Methods of the public abstract class Component.

An alternative and possibly more convenient way of receiving mouse events is to use a mouse adapter, with the public abstract class MouseAdapter implementing MouseListener.

Unlike with most events, mouse events are delivered to a component's listeners before the operation takes place. This gives the listeners an opportunity to consume the event by using *consume ()*, one of the consume methods of the public abstract class InputEvent.

(3) ***Mouse Event Flow***

Figure 5-1 shows how mouse events typically flow through the system. First, the event is fired by a component peer in response to the user's interacting with the mouse. This event is posted on the event queue, implementing by the public class EventQueue. When the event makes its way to the front of the queue, it is given to the component via its *dispatchEvent ()* method, one of the event methods of the public abstract class Component. The main purpose of this method is to forward the event directly back to the component peer if the mouse event type is not enabled or if there are no mouse listeners. Otherwise, *dispatchEvent ()* calls *processEvent ()*, which in turn calls different methods depending on the event type. Since this is a mouse event, *processMouseEvent ()* is called. The main purpose of this method is to notify the mouse event listeners. Finally, if the event has not been consumed, it is forwarded back to the originating component peer.

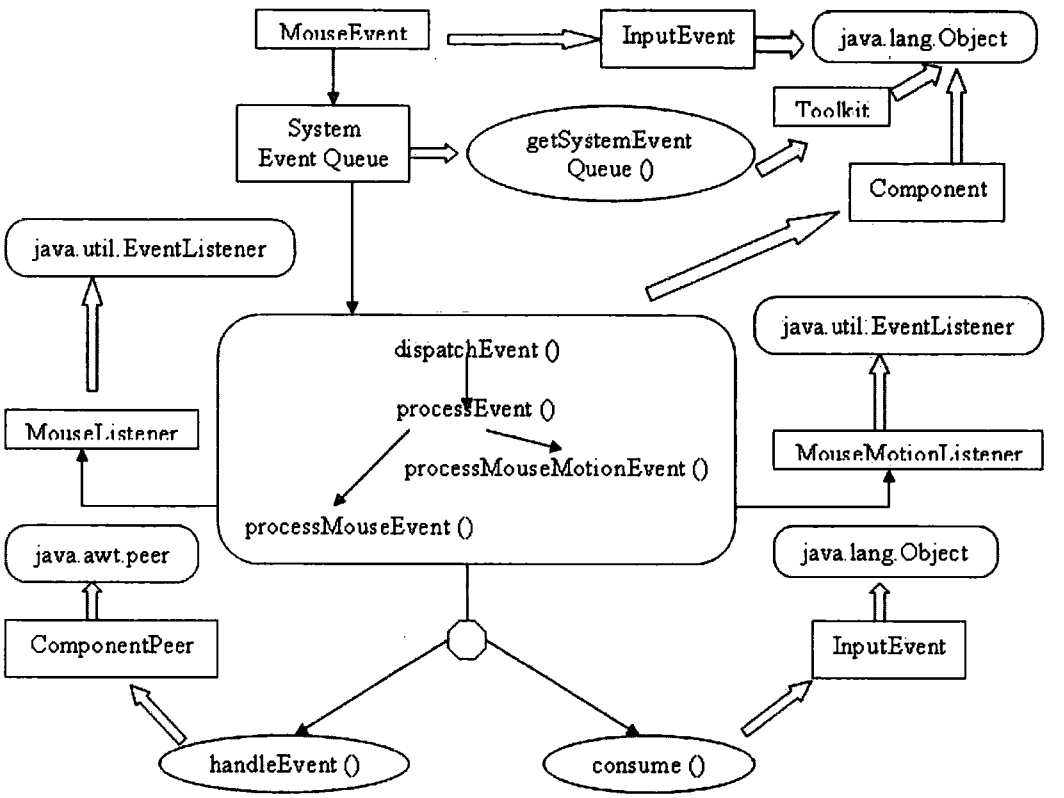


Figure 5-1: Mouse Event Flow

A component can override *processMouseEvent ()* to process mouse events before they are delivered to its listeners. The overridden method should call *super.processMouseEvent ()* to ensure that events are dispatched to the component's listeners.

From the example above, therefore, we can see that numerous of classes and functions need to cooperate together to respond to a mouse event. Input-output pairs and semantic bindings guarantee the integration and the relationship among them forms a "process net", where transmitting information forms a flow as well. Moreover, we can also see that the step-by-step composition of services, from fundamental classes and functions up to complex composites.

5.1.2 General Relationship between Services

Figure 5-2 shows the general relationship between services [7]. Service x may have relationship with service y and z, and y may also have relationship with z. Relationship can be either direct or indirect. So if service y has no direct relationship with z, but because both of them interact with x at some stage, there is at least indirect relationship between y and z.

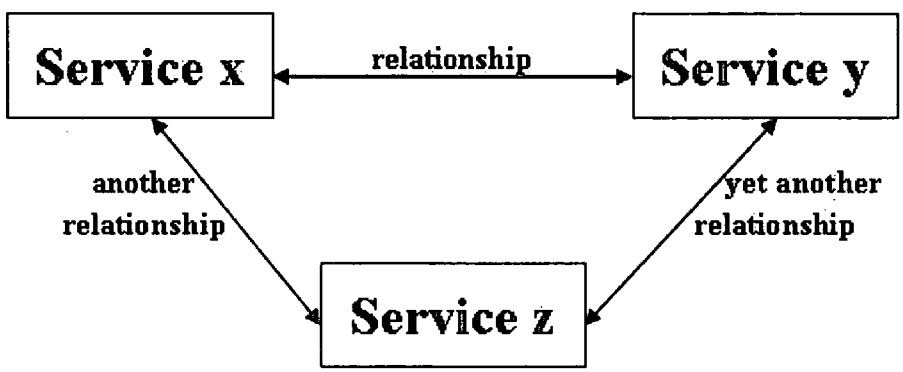


Figure 5-2: Basic Interrelationship between Services

If we use the graph theory to describe this kind of relationship between services, it can denoted as g(v, e), where $v=\{S_i, i=1, 2, \ldots, n\}$, $e=\{(S_i, S_j) \mid S_i, S_j \in v\}$, which means the vertex of the graph corresponds to a service and the edge representing the relationship between them. Because this service, composed from three atomic services as illustrated in Figure 5-2, is also a component of the top-most service, the composed graph is, thus, a subset of the complete graph, denoting as $g(v, e) \subseteq G(V, E)$. However, this top-most service is only a fraction of the entire service world which has an amount of 2^G services, the power set of one single complete service. Any service, therefore, is an element of it, as represented by $s \in 2^G$. The entire services can thus be interpreted in natural languages with the help of adding semantics to their descriptions. From this point of view, it is critical and feasible to start composing services according to a criterion, which is atomic service in our research work.

5.2 Criteria of Atomic Service

5.2.1 Motivation: Why Atomic?

As defined in Chapter 3, Atomic Service is a simple and undecomposable service with a lifetime of a single request. The proposition of atomic service aims to provide a regulation base on which services can be defined and invoked coequally.

Current composition methods, however, have no clue to select which level of services to start the composition process. In fact, those methods are based on the mechanism of "one-off requirement-matchable" fashion, which means the composition only concentrates on finding a suitable service for this particular session. They do not help re-selecting the same service in the future as there is no service module to follow. Moreover, they are not able to create a standard composition level as services may be selected at random level every time. Therefore, we suggest that services be composed from fundamental components, which are atomic services, thus the service modules produced through each composition session can be reused in future.

5.2.2 Standard of Recognition

Judging whether a service is atomic depends on its characteristics and the current environment as well. First of all, it should satisfy the definition of atomic service, which is not decomposable with single request. Moreover, the status of a service may change due to various environments. For example, relocating a desk during decoration can be viewed as an atomic service, as comparing to all the other tasks, it is simple enough to be specified as a one-shot action. However, in a cabinetmaker's point of view, to manufacture a desk is far more than a simple atomic service as it can be decomposed to many steps.

It is to say, therefore, the status of a service needs to be classified according to the environment and requirement level, rather than being judged at first glance.

5.3 Performance and Comparison

The demonstration system introduced in the previous chapter is based on the proposition of the modified service structure aided by the concept of Atomic Service, the composition conventions and the scheduling mechanism. Therefore, the contribution of the system can be view in three different aspects.

5.3.1 Service Structure and Description

Although the widely-used OWL-S is able to use the service profile to describe a service and its functionality, it lacks a practical expression in terms of how a set of selected fundamental services are put together and executed, in another words, being composed and scheduled. However, our service structure based on the introduction of atomic service is capable of inducing the scheduled service composition.

One improvement is to add an agent, which specialises in dealing with the scheduling issue, to the structure of atomic service. It is recorded in the service description document under decomposition and subsequential service. This is to say, the execution sequence of all the fundamental services is defined at the stage of analysing and decomposing a topmost service requirement, and it is quite efficient that such document analyse only needs to be done once at the very beginning of the whole service procedure. Afterwards, the description will be referred as services are composed and scheduled.

5.3.2 Service Composition

Compared to current Web Services composition approaches such as BPEL and OWL-S, the advantages of using the composition method based on atomic service can be viewed in the following perspectives.

(1) Scalability

Existing composition methods are not capable of dealing with massive service integration when the number of services accumulates to some extent. This is simply because the augmentation of XML files which BPEL and OWL-S are based on will hamper the composition process [31]. However, the atomic service-based composition approach can be illustrated in corresponding graphs and will thus convert the service composition into the graph integration issue. This will inevitably enhance the scalability of service composition.

(2) Depth

The realisation of a requirement relies on the allocation of various resources eventually. In this case, atomic service-based composition features a prompt and reliable way of collecting necessary resources because of the fundamental status of atomic service. The execution of each atomic service is provided by a class or method in the lib file, a certain amount of memory usage and CPU time.

(3) Efficiency

Unlike other composition approaches, the execution time property in the service description facilitates that the running time of each service is monitored. It shortens the waiting time of available services by setting up the response mechanism for each atomic service. The exception handling mechanism also makes the composition process more efficient by invoking alternative services in case of the occurrence of any exceptions. Most importantly, as mentioned above, the one-off analyse of the service description makes the following service composition more fluently, with the major time-consuming part falling on the transmission of response signals and execution of each service.

5.3.3 Service Scheduling

The performance of the scheduling system, which manages the running sequence and time control of services, is yet able to be measured according to any quantitative

criterion at this stage. However it can still be evaluated according to the comparison between scheduled result and the expected one in certain perspective.

For example, it can be examined whether the composite service after being scheduled is able to meet the original requirement. This is able to be verified by feedback from requestors. If it can somehow fulfil the task, the percentage of the achievement is an important index, as it can evaluate both the performance of scheduling system and the quality of services. This is called composition correctness review and can be viewed in Figure 5-3, which gives the comparison among three major service scheduling approaches: the proposed atomic service-based scheduling method, middleware-based scheduling like OMII and the job scheduler linked with Globus Toolkit 4.

Evaluation Aspects / Scheduling Approaches	Service Catalogue	Service Searching Ability	Time Cost	Consistency Check	Composition Correctness Review
Atomic Service-Based Scheduling	Service Pool	Strong	Average	EHS	✓
Middleware-Based Scheduling	✓	Medium	High	✓	✓
Job Scheduler	✓	Strong	Low	N/A	✓

Figure 5-3: Comparison on Current Service Scheduling Approaches

As seen in the figure, all scheduling methods need to be based on a service catalogue which enables service selection. Our atomic service-based scheduling uses service pool as the catalogue. It has a strong service searching ability as it is based on fundamental functional services which are name-searchable and directly executable. It is not the quickest scheduling method as the service description needs to be thoroughly analysed at the first stage before selected services can be scheduled together according to the description. However, the average time cost can be lower down with the expansion in the scale of services. Moreover, atomic service-based scheduling enables algorithm selection which will produce efficient scheduling under

different circumstances. Last but not the least, compared to other approaches, the Exception Handling System (EHS) it uses facilitates the check-back on the consistency of service and will deal with any exceptions if a selected service cannot produce the expected result.

5.3.4 Equipment Requirements

This demonstration system does not have high hardware requirements such as lengthy CPU time or large memory usage in order to perform the procedures of service composition and scheduling at this stage. However, it has currently got a small size of base to store the service pool which is only suitable for simple service compositions. It needs to be expanded gradually so as to take in more complex service requirements and decompose them into storable service components.

5.4 Summary

Based on the introduction of atomic service and service scheduling method, along with the demonstration system illustrated in the previous chapter, this chapter starts with presenting the theoretical grounds for using atomic service as a core part of service scheduling and composition. The interrelationship among basic classes facilitates the composability of one atomic service with others. After that, the evaluation of service composition and scheduling method is presented through the comparisons against the current approaches. The conclusion can be drawn that atomic service-based composition and scheduling does have certain advantages in terms of service scalability and consistency.

Chapter 6 Conclusions

The above all chapters have presented some key issues within current Web Services research, particularly emphasised the importance of extending existing service description approaches, which serves as the base of interactions like composition and scheduling.

6.1 Overview

The evolution of Internet technologies in these years makes WWW not only a massive carrier of ever-increasing information, but facilitates people to make interactions online as well. The Internet thus becomes a medium where all of its users are able to share network resources and meet their requirements. On the other hand, it is ideal to make machine-to-machine interactions possible and the performance for using machines to search for desired service is highly expected. Therefore, a lot of research has been carried out in Web Services in order to make every effort to meet requirements from service requestors by using current online services, resources and technologies. The first chapter of this thesis introduced the background of Web Services and the infrastructure of Semantic Web.

The second chapter went over some major issues within Web Services nowadays. The two main issues discussed are service description and composition. Service-Oriented Architecture (SOA) provides the three entities of Web Services and defines the approaches for communication between each other. Web Ontology Language (OWL), which is used to present the meanings of relationship between vocabulary terms, is based on description logic and thus disabled to express rules. In case of existence of unspecified constraints in making queries in the Semantic Web, the OWL-supported knowledge based may not turn out to have proper results. Web Services Description Language (WSDL) plays an important part in service publish and service binding for service provider and requestor respectively. The lack of semantics in describing

service forces users of WSDL to improve it with added properties or other customisations.

Moreover, service composition has received lots of interest because of the reusability of service modules. Several approaches have been put forward, including BPEL4WS, OWL-S and software components etc. There is yet a standardised way to perform service composition, leaving all current methods some disadvantages more or less. Although the relevant software reuse has not turned out to be a success in recent years, service composition is still a promising perspective and business integrations largely rely on it.

Chapter 3 is the core part of this thesis where individual ideas were proposed. First of all, we brought forth the concept of Atomic Service, a simple service with a lifetime of a single request. Services all around are glued eventually by atomic services and can be decomposed step by step till the stage of atomic service. The purpose for proposing this concept to provide a criterion on the level of description and composition, making all issues centralised at atomic service, which is the fundamental component of all software pieces. Then, the architecture of describing services was put forward in the wake of the birth of atomic service. Since WSDL fails to indicate which service is to be invoked at what stage, the proposed model adds properties like decomposability (whether the service being processed is able to be further decomposed) and subsequence (indexing the following service in order to make all services running in a row). Meanwhile, according to the ontological interrelationship between service and graph, the description of services can be correspondingly converted to the representation of graph, especially conceptual graph.

Given the description of atomic services and service structure, the realisation of a service at high level can be operated in two directions: top-down convention and bottom-up convention. The first convention aims to setting out from the service requirement, through gradual decomposition and refinement to achieve the goal, while

the second convention does in the opposite way. Starting from atomic services, it levels-up by integrating other atomic services to composite services in order to form a service module. Appropriate service modules will then be synthesised to meet the original requirement.

One of the key issues remaining in the process of composition is how to combine relevant services and how to control the sequence. It is suggested that services can be integrated when their interfaces are matchable, not only the type and amount of input and output data should be matched, but the semantic level of composition should also be reached. On the other hand, a scheduling policy has been carried out with respect to monitoring composition processes. The scheduling model indicates the framework of describing scheduling language. The process of composing services refers to the three sessions, each of which explains the mechanism of service operations. The Web Services Scheduling Language (WSSL), along with two scheduling fashions, can cover all possible service composition styles and are able to direct the whole process.

After addressing the service realisation at both functional side and business level, the discussion on system design and the implementation of a demonstration system were presented in Chapter 4. It first discusses the three scheduling sessions which deal with the dynamic aspects of how atomic service are communicated through composition, and then turns the concept of atomic service into practicality, aiming to feature the process of service selection and composition according to its description. The scheduling model, discussed in Chapter 3, is represented through building up the whole composition procedure, along with the entire service flow.

In order to demonstrate the feasibility of applying composition and scheduling methods on atomic service, Chapter 4 put up an investigation on an example from lib file and proved that each atomic service more or less has some relationship with other services during communication and interaction. It then made a deep research on the feasibility of proposing the concept of atomic service and criteria of judging an atomic

service. Finally, it made a qualitative analysis of the performance of scheduling system by means of comparing the atomic service-based scheduling with other contemporary approaches.

6.2 Discussion

This thesis mainly discusses about the following three issues in Web Services: description, composition and scheduling. Approaches of describing services have been proposed throughout the years by using OWL for ontology and structure, WSDL for language, SOAP for message exchange and BPEL4WS (formerly DAML+OIL) for business integration etc. However, apart from WSDL and SOAP, there are still lots of description methods outside the recommendation of W3C. The deficiencies of descriptions themselves and the non-standardisation make it difficult for services to be explicitly declared. Therefore, researches on enhancing the ontology and improving the service description have been carried out. Nevertheless, SOA is widely accepted to be adopted as the fundamental structure of service base, enabling both service providers and service requestors to interact with each other with the aid of service registry. But the shortage of using current WSDL to describe Web Services lies in the fact that no information is specified for operation invocation within the process and the consumption of using service is not well-timed according to the vague description. W3C has been, therefore, trying to add process information to indicate the above problem to the description language in the perspective of choreography. Accordingly, the description model we proposed aims to specify this part of property, making it clear that services are well-defined in terms of process and added properties can facilitate further operations such as composition on services.

Using the notion of software reuse for reference and aiming the target of business integration, how to utilise the combination of existing services to make vast performance is under delicate research. One direct thought is to analyse the service requirement, extract the key information from the description, decompose it into

simple services and convert them into machine-processable programs. The whole procedure involves studying the semantic structure of the requirement as expressions in natural language cannot be recognised directly by machines. The decomposition action is mainly performed manually as currently there is no suitable decomposer capable of parsing the requirements and converting them to machine-understandable language. The proposed top-down convention focuses on a decomposition mode which is a further step into bringing automation to the pure manual decomposition, as the applied conceptual graph theory in representing service description can help dealing with service operations. Conceptual graph is based on semantic network and is, importantly, both human-readable and computation-tractable [42]. Therefore, the decomposability of a service can be performed by machines according to whether a corresponding graph can be resolved or not.

On the contrary, composition issues commencing from fundamental services up to the top level have drawn a lot of interest. Several approaches of composition involve BPEL4WS, OWL-S, and software components etc. BPEL is widely used in business integrations as companies like IBM, Microsoft, BEA and Oracle all base on it for orchestrating end-to-end business processes. By adopting Web Services as its external communication mechanism, BPEL uses WSDL for describing incoming and outgoing messages, which make important parts in business transactions. On the other end of service domain, decomposition focuses on how to weave individual functional services together to have complex performances. The basic consisting part is atomic service, which is the cornerstone of composition. Attaching operation as a property to atomic service makes it explicitly what kind of actions can be imposed on the service and the detailed description of parameters indicates that both the value and the type of data should be matchable in order to be integrated. After combining all relevant services together, the implementation is then scheduled into a workflow following the data transmission. The scheduling mechanism, therefore, is proposed in order to form a smooth flow and make services seamlessly coupled.

Compared to issues like service description and composition, scheduling has not drawn enough attention so far. However, it is credited to the scheduling mechanism to solve problems like service selection and arrangement. Scheduling offers canonical order for service operation and links up with exception handling process in case of the occurrence of unexpected errors. The implementation or individual services are now arranged in the order which coincides with the process flow. The operation imposed on the description of each service offers a clue of the following step and the scheduling system will take the corresponding action according to the description.

6.3 Criteria for Success

This thesis has mainly covered these following research issues:

1. Discussions on current service description, composition and scheduling approaches, which have been done in the literature survey, especially in sections 2.2, 2.4 and 2.5;
2. Proposals of the modified service description structure based on the introduction of the concept of Atomic Service have been drawn in section 3.4;
3. Introduction of the service composition and scheduling issues based on Atomic Service which is put forward in section 3.6;
4. Illustration of the Atomic Service-based scheduling and composition through the implementation of a demonstration system in Chapter 4;
5. Reasoning and evaluations on service structure and approaches of service composition and scheduling in Chapter 5.

6.4 Conclusions and Future Work

This thesis demonstrates the service description based on the concept of atomic service by means of importing the atomic service into composition and scheduling. The structure of describing a service refers to OWL-S by specifying relationship with composition, indicating the decomposability of the service. Moreover, the proposed description highlights the mechanism of scheduling, claiming the state and prerequisite which the service should meet in order to be integrated. Meanwhile, it

involves the conditions based on composition rules, which should be followed by services to be seamlessly coupled.

In terms of composition, atomic service-based pattern, which levels the scale of service from bottom, stands in contrast to the pattern in which a service requirement is decomposed from top level to foundation. Scheduling plays an important part in both occasions. It arranges the integration style and order in the bottom-up pattern, while in the top-down convention, it shows the direction of how services should be decomposed according to the requirement and the distribution of possible components.

Despite numerous researches which have been carried out on Web Services, there are still some open problems yet to be discussed and resolved. In functional composition, the correctness check remains unsolved. Composite services derived from gluing atomic services should be fully matched to the original requirement. Thus, checking the identical degree between composite service and requirement becomes critical in implementation. Otherwise, in case of unavailability of completely-matchable service, the distance between the alternative service and the requirement needs to be calculated in evaluating the performance.

On the other hand, the issue of making decomposition thoroughly automatic is still pending to be solved. Some automatic work can be done when applying a semantic-based requirement recogniser, but it is only restricted to a well-defined requirement which can meet the semantic structure. Any loss in representing a user requirement may result in loose couple composition and return unexpected results.

The next step of our research will focus on expanding service profiles and reasoning service operations. The service profile should involve both static descriptions and dynamic operations. Some of the already-defined actions in service description like decomposition need to be automatically linked up with dynamic operations to be

triggered. Reasoning will cover the examination into the availability of existing services and the feasibility of replacing the original-targeted service with its alternative. We also hope to import the mechanism based on atomic service into business process to facilitate service matchup and use the scheduling language to manage business integration. There is still a lot of work to be done to ideally link functional services and business services.

References

[1] Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju, *Web Services – Concepts, Architectures and Applications*, Springer-Verlag, 2004.

[2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana, "Business Process Execution Language for Web Services, Version 1.1," IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, May 2003.

[3] Daniel Austin, Abbie Barbir, Christopher Ferris, Sharad Garg, "Web Services Architecture Requirements," W3C Working Group Note 11 February 2004.

[4] Keith Ballinger, David Ehnebuske, Christopher Ferris, Martin Gudgin, Canyang Kevin Liu, Mark Nottingham, Prasad Yendluriet, "WS-I Basic Profile Version 1.1," The Web Services-Interoperability Organization (WS-I), 2004.

[5] Tim Berners-Lee, "WWW past & future," Royal Society, W3C, 2003.

[6] Jason Bloomberg, Ronald Schmelzer, *Service Orient or Be Doomed!: How Service Orientation Will Change Your Business*, Wiley, March 10, 2006.

[7] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, David Orchard, "Web Services Architecture," W3C Working Group Note 11 February 2004.

[8] Jeffrey C. Broberg, "Glossary for the OASIS WebService Interactive Applications (WSIA/WSRP)," OASIS, 2002.

[9] Luis Felipe Cabrera, Christopher Kurt, Don Box, "An Introduction to the Web Services Architecture and Its Specifications," Version 2.0, Web Services Technical Articles, Microsoft Corporation, October 2004.

[10] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1," W3C Note 15 March 2001.

[11] Moon Jung Chung, Woongsup Kim, Ravi Gopalan, Hong Suk Jung, Hyun Kim, "Service Model for Collaborating Distributed Design and Manufacturing," in

Proceedings of the WWW 2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web, New York, NY, 2004.

[12] Jos de Bruijn, Dieter Fensel, Uwe Keller, Rubén Lara, "Using the Web Services Modelling Ontology to enable Semantic eBusiness," *Communications of the ACM*, Vol. 48, Issue 12, pp. 43-47, December 2005.

[13] Stefan Decker, Sergey Melnik, Frank Van Harmelen, Dieter Fensel, Michel Klein, Jeen Broekstra, Michael Erdmann, Ian Horrocks, "The Semantic Web: The Roles of XML and RDF," *IEEE Internet Computing*, Vol. 4, No. 5, pp. 63-74, September/October 2000.

[14] The Gene Ontology Consortium. "Gene ontolgy: Tool for the unification of biology," *Nature Genetics*, 25(1), pp. 25-29, 2000.

[15] Roman Ginis, K. Mani Chandy, "Service Composition Issues for Distributed Business Processes," *ICWS*, 2003, pp. 27-33.

[16] The Globus Toolkit, "GT 4.0 Execution Management Glossary," http://www.globus.org/toolkit/docs/4.0/execution/wsgram/WS_GRAM_Glossary.html

[17] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web Services Architecture," *IBM Systems Journal*, Vol. 41, No 2, 2002.

[18] Mingyang Gu, "*Conversational Case-Based Reasoning in Software Component Reuse,*" Trondheim, Norway, reported in June 2004.

[19] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework," W3C Recommendation 24 June 2003.

[20] Jeff Heflin, Raphael Volz, Jonathan Dale, "Requirements for a Web Ontology Language," W3C Working Draft 07 March 2002.

[21] Jane Hunter, Sharmin Choudhury, "Semi-Automated Digital Preservation System based on Semantic Web Services," in *Proceedings of the 4th ACM/IEEE-CS joint conference on Digital libraries*, 2004, pp. 269-278.

[22] IBM, "Standards and Web Services," http://www-128.ibm.com/developerworks/webservices/standards/.

[23] Charlotte Jenkins, Mike Jackson, Peter Burden, Jon Wallis, "Automatic RDF

Metadata Generation for Resource Discovery," *Computer Networks: The International Journal of Computer and Telecommunications Networking archive*, Vol. 31, Issue 11-16, pp. 1305-1320, May 1999.

[24] Takahiro Kawamura, Jacques-Albert De Blasio, Tetsuo Hasegawa, Massimo Paolucci, Katia Sycara, "Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Registry," *Lecture Notes in Computer Science*, pp. 208-224, 2003.

[25] Michael Klein, Birgitta König-Ries, Philipp Obreiter, "Stepwise Refinable Service Descriptions: Adapting DAML-S to Staged Service Trading," in *Proceedings of the First Intl. Conference on Service Oriented Computing*, Trento, Italy, 2003, pp. 178-193.

[26] Frank Manola, Eric Miller, Brian McBride, "RDF Primer," W3C Recommendation 10 February 2004.

[27] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, Katia Sycara, "OWL-S: Semantic Markup for Web Services," W3C Member Submission 22 November 2004.

[28] Mkhhail Matskin, Jinghai Rao, "Value-Added Web Services Composition Using Automatic Program Synthesis," Lecture Notes in *Computer Science*, Vol. 2512, Revised Papers from *the International Workshop on Web Services, E-Business, and the Semantic Web*, pp. 213-224, 2002.

[29] E. Michael Maximilien, Munindar P. Singh, "Conceptual Model of Web Services Reputation," *ACM SIGMOD Record*, Vol. 31, Issue 4, pp. 36-41, December 2002.

[30] Carolyn .McGregor, "A Method to extend BPEL4WS to Enable Business Performance Measurement," *ICWS*, pp. 46-54, 2003.

[31] Nikola Milanovic, Miroslaw Malek, "Current Solutions for Web Service Composition", *IEEE Internet Computing*, 1089-7801/04, 2004.

[32] Soraya Kouadri Mostéfaoui and Béat Hirsbrunner, "Towards a Context-Based Service Composition Framework," in *Proceedings of the International Conference on Web Services, ICWS*, Las Vegas, Nevada, 23-26 June 2003

[33] Yefim V. Natis, "Service-Oriented Architecture Scenario," Gartner, AV-19-6751, 16 April 2003.

[34] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara, "Semantic Matching of Web Services Capabilities," in *Proceedings of International Semantic Web Conference (ISWC)*, Sardinia, Italy, 2002.

[35] Paulo F. Pires, Mário R. F. Benevides, and Marta Mattoso, "Building Reliable Web Services Compositions," Lecture Notes in *Computer Science*, Vol. 2593, Revised Papers from *the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, Springer-Verlag, 2002, pp. 59 - 72.

[36] Marco Pistore, F. Barbon, Piergiorgio Bertoli, D. Shaparau and Paolo Traverso, "Planning and Monitoring Web Services Composition," ICAPS'04 Workshop on Planning and Scheduling for Web and Grid Services, 2004.

[37] Chris Preist, "A Conceptual Architecture for Semantic Web Services (Extended version)," shorter version HPL-2004-214, *International Semantic Web Conference*, Hiroshima, Japan, 8-11 November 2004

[38] Will Provost, "UML for Web Services," O'Reilly xml.com, August 05, 2003, http://webservices.xml.com/pub/a/ws/2003/08/05/uml.html.

[39] Jinhai Rao, Peep Küngas and Mihhail Matskin, "Application of Linear Logic to Web Services Composition," in *Proceedings of the 1st International Conference on Web Services*, Las Vegas, USA, June 2003.

[40] Douglas C. Schmidt, "Why Software Reuse Has Failed and How to Make It Work for You," *C++ Report magazine*, January 1999.

[41] Wei Song, Ming Zhang, *A First Step towards the Semantic Web*, Higher Education Press, 2004.

[42] John F. Sowa, *Conceptual Graphs*, ISO/JTC1/SC 32/WG2 N 000, April 2001.

[43] Biplav Srivastava, Jana Koehler, "Planning with Workflows - An Emerging Paradigm for Web Services Composition," ICAPS Workshop on Planning and Scheduling for Web and Grid Services, 2004.

[44] Biplav Srivastava, Jana Koehler, "Web Services Composition – Current Solutions and Open Problems," ICAPS Workshop on Planning for Web Services, 2003, pp.

28-35.

[45] Michael Stevens, "'Service Oriented' Architectures, Part 1," SSA Research Note SPA-401-068, 12 April 1996; "'Service Oriented' Architectures, Part 2," SSA Research Note SPA-401-069, 12 April 1996.

[46] Brian E. Travis, Mae Ozkan, *Web Services Implementation Guide Volume 1: Getting Started*, Architag Press, 2002, p. 276, pp. 283-284.

[47] UDDI.org, "UDDI Executive White Paper," November 14, 2001.

[48] W3C, "Web Services Activity," http://www.w3.org/2002/ws/.

[49] Yifei Wang, Hongbing Wang, Xun Xu, "Web Services Scheduling: Binding the Cost with the Time," in *Proceedings of the First International Conference on Semantics, Knowledge, and Grid (SKG 2005)*, IEEE, 0-7695-2534-2/05, 2006.

[50] Petia Wohed, Wil M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede, "Analysis of Web Services Composition Languages: The Case of BPEL4WS," in *Proceedings 22nd International Conference on Conceptual Modelling (ER)*, Chicago IL, USA, 2003, pp. 200-215.

Appendix Thesis Glossary

Atomic Service: a simple and undecomposable service with a lifetime of a single request

Composition: the task of putting together atomic and/or composite services to perform complex tasks

Critical Point: dependent relationship between services

DAML-S: a semantic markup language to describe service and ontology built on top of DAML+OIL

Decomposition: the procedure of "dissecting" a service into its sub-services with the aim of being easily processed

OWL-S: OWL-based Web Services Ontology

RDF: Resource Description Framework, a general method of modeling knowledge

Registry: a directory that contains information about available services

Scheduling: the planning of services or operations

Semantic Web: project of the W3C in which automated methods based on quality metadata are envisaged to replace much human searching of the web

Service Requestor: client of a service element

Service Provider: an entity that provides services to other entities

SOA: a service-oriented architecture is a collection of services that communicate with each other

UDDI: a XML-based protocol that provides a distributed directory that enables businesses to list themselves on the Internet and discover other services

Web Services: the programmatic interfaces which make application to application communication available

Workflow: order in which specific work is performed

WSDL: the language based on XML to describe and locate Web Services and specifies the methods which are used to get access to services

WSSL: a self-developed Web Services Scheduling Language