NOTE TO USERS

uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Deyang Liu
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

A Selection Algorithm for Composing Web Services in Sensor Network

TITRE DE LA THÈSE / TITLE OF THESIS

A. El Saddik
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Ramiro Liscano

Chung-Horng Lung

Gary W. Slater
LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

A selection Algorithm for Composing Web Services in Sensor Network

By

Deyang Liu

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements for the degree of
Master of Applied Science
In
Computer Science

Ottawa-Carleton Institute of Computer Science
School of Information Technology and Engineering

Faculty of Engineering
University of Ottawa

Your file Votre référence
ISBN: 0-494-11329-4
Our file Notre référence
ISBN: 0-494-11329-4

Abstract

Web services and sensor networks have been undergoing tremendous growth during the past few years. Integrating web services technology to sensor networks domain offers many advantages. For example, the existing Internet protocols can be used instead of designing proprietary ones, 3rd party can deliver value-added web services to end users easily, etc. However, wireless sensor networks are constrained by limited system resources, such as memory, power, and computation capacity.

In this thesis, we propose a web service selection algorithm by introducing QoS-based *cost function* concept. The mechanism we used is choosing only a few web services from each service community according to their *cost function.* Then we compose them together to find the optimal composite service. The proposed selection algorithm consumes only limited system resources and has the characteristics of efficiency, dynamics, and fault tolerance. It guarantees the composed web service has the optimal performance in terms of cost function.

Acknowledgements

First and foremost, it has been an honor and privilege to have Dr. Abdulmotaleb El Saddik as my supervisor. His immense talent and enthusiasm for research greatly inspired me. I have benefited a great deal from his constant encouragement, and the breadth and depth of his knowledge.

My work has been benefited from and influenced by discussions with the members of Instantel Project. For these discussions, I express my sincere gratitude to Dr. Ramiro Liscano. My gratitude also goes to my colleague ajith kamath, Emil Sadok, and Li Wu.

In addition, I would like to thank the members of the Multimedia Communication Research Laboratory for their support and friendships. Especially, I want to thank Yong Zhong for her help.

Finally, I am indebted to my wife, Ying Shao, for being a constant source of comfort, support, and encouragement. I cannot imagine how to go through the graduate studies without her support. My son, Bryan Liu, has brought me a great deal of joy and made my life much more colorful.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Program Interface
BPEL4WS	Business Process Execution Language for Web Service
BPML	Business Process Markup Language
BPMN	Business Process Modeling Notation
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
FTP	File Transfer Protocol
HTTP	HyperText Transfer Protocol
MAC	Medium Access Control
NCAP	Network Capable Application Process
OWL-S	Web Ontology Language for Service
QoS	Quality of Service
RMI	Remote Method Invocation
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WSCI	Web Service Choreography Interface
WSDL	Web Service Description Language
WSFL	Web Services Flow Language
XLANG	Web Services for Business Process Design
XML	eXtensible Markup Language
XML-RPC	XML Remote Procedure Calls

Chapter 1

Introduction

1.1 Motivation

Due to the nature of self-contained, loosely coupled, platform and language independency, web services technology is becoming increasingly popular in many areas [40]. Web services are modular web applications that can be published, located, and invoked across the web by other web applications or services. The modularity and flexibility of web services make them ideal for application integration. By adopting open standards (such as SOAP and WSDL), enterprises can easily integrate in-house service components with external web services to provide a value-added business process.

Meantime, sensor networks have been undergoing tremendous growth during the past few years. Many new applications have started to emerge. Because of the recent advances in wireless communications and electronics, nowadays, the focus is more on wireless, distributed sensor networks [56].

Integrating web service technology to sensor networks domain offers many advantages. For example, the existing Internet protocols can be used instead of designing proprietary ones; 3rd party can deliver value-added services to end users easily based on existing standards; people can acquire and manage sensory data from all around the world via Internet. On the other hand, wireless sensor networks are constrained by limited resources, such as onboard memory, computation capacity, and battery power. This makes it quite challenging for integrating these two domains together.

The purpose of this thesis is to investigate existing web services composition techniques and propose two web service selection algorithms that are efficient and resource-effective. The proposed service selection algorithms can be used in the area of sensor networks.

1.2 Research Objective

The current sensor networks are designed for specific applications, with data communication protocols strongly coupled to the application [45]. We argue that the future sensor networks will compose of heterogeneous devices supporting a large range of applications. With the widespread of Internet, accessing and managing sensory data become possible and imperative. The mature web service technology provides a way to establish web service based architecture for sensor networks.

By integrating web service technology to sensor networks domain, one can apply some *web services composition technique* to combine several web services together to provide a value-added web service. However, the inherent characteristics of sensor networks, such as limited storage and computation resources and battery power, pose great challenges to the design of web services composition technique. The current web services composition solutions, as summarized in references [1, 3, 5], do not consider the resources limitation requirements.

Web services composition is a complicated process. It consists of the following four steps: service discovery, service selection, service binding, and service execution. In our research, we focus on the service selection step, i.e. selecting the most appropriate web services for composition. In order to do that, we introduce the *cost function* concept that can be used to distinguish and choose web services.

Our research objective is to design and verify a resource-aware web service selection algorithm that can be used in sensor networks domain as well as other areas. The proposed selection algorithm consumes only limited system resources and has the characteristics of efficiency, dynamics, and fault tolerance. It guarantees the composed web service has the optimal performance in terms of cost function. By using the proposed service selection algorithm, web services can be dynamically composed into value-added applications.

1.3 Research Contributions

In this thesis, we present a novel approach for web services composition that can be used in sensor networks. A number of contributions are listed below.

- Analysis of existing web service composition languages and techniques
- Analysis of the Quality of Service parameters for web service
- Introducing the *cost function* concept based on some QoS parameters and user's preference; cost function value can be used to distinguish web services among a service community
- Proposal of a service selection algorithm for choosing the best elementary web service
- Proposal of a service selection algorithm that can be used for optimizing the composite service; the proposed algorithm consume only limited computation and storage resources
- Testing the performance of the proposed selection algorithm with a large amount of simulation

1.4 Organization of Thesis

We have stated our research motivation, objective, and contributions. The remaining part of the thesis is organized as follows.

Chapter 2 provides the background knowledge of web service and web services composition. We first introduce the concept of web service, the protocols used, and their architectures. Then the current web services composition languages and techniques are discussed.

Chapter 3 gives a brief overview about sensor networks, their main characteristics, protocol stack, and applications.

Chapter 4 presents our proposed web service selection algorithms. First of all, we analyze the Quality of Service parameters for web services. After that, we define the concept of *cost function*. Based on cost function, two web service selection algorithms are proposed, one for elementary web service and the other for composite web service.

We concentrate more on the selection algorithm for composite web service.

Chapter 5 shows our implementation and experiment results.

In **Chapter 6**, we first analyze the overall experiment results listed in chapter 5. Then the fault tolerance issues are thoroughly discussed. Finally, we evaluate the complexity of our proposed algorithms.

Chapter 7 summarizes and concludes the thesis. It also provides recommendations for future research.

Chapter 2

Web Service and Web Services Composition

Web services have become an emerging and promising technology for distributed computing and e-businesses. In this chapter, we first introduce what are web services, the standards used in web services, and the web services architecture. Then we give the concept of web services composition and related techniques. This chapter and chapter 3 are intended to provide the reader with the background information needed to understand our research.

2.1 Web Service Overview

> *Today, the principal use of the World Wide Web is for interactive access to documents and applications. In almost all cases, such access is by human users, typically working through Web browsers, audio players, or other interactive front-end systems. The Web can grow significantly in power and scope if it is extended to support communication between applications, from one program to another.*
>
> *- From the W3C XML Protocol Working Group Charter[57]*

A web service is a self-contained, language neutral, platform independent, and loosely coupled application that can be described, published, located, and invoked over the Internet (See Figure 2.1). Unlike traditional web-based applications, which deal with communication between humans and programs, web services can be used to link application-to-application communications within organizations, across enterprises, and across the Internet automatically. References [32, 40] provide an overview about the web services technology and references [33, 43] give tutorial-level information for developing Java-based web services.

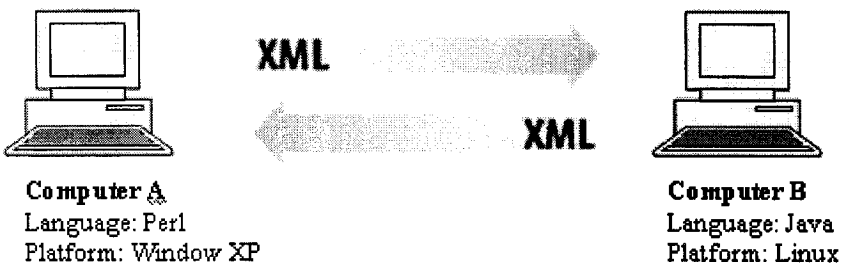

Figure 2.1 A basic web service example [31]

Web services are a new breed of web applications. They are the fundamental building blocks in the move to distributed computing on the Internet. Most of the existing distributed protocols like CORBA, RMI, COM, DCOM, and socket etc have the problems of tightly coupling, language dependency, and vendor dependency. Web services overcome all these problems by using XML-based standards and protocols. Like component-based development, web services provide black-box functionality that can be reused without worrying about how the service is implemented. The functions that web services can perform range from simple requests to complicated business and scientific processes.

Web services reflect a new Service-Oriented Architecture (SOA) approach. They represent an evolution from object-oriented systems to service-oriented systems. In the web services society, everything is built as a service, so reusability and scalability are greatly enhanced. Web services are accessed via ubiquitous web protocols such as HTTP and an XML data representation.

Listed below are some potential benefits expected from the web service technology.

- ***Loose coupling:*** In web services model, all components are loosely-coupled. This implies that a change in the structure and implementation of the services does not require any change in the applications using them. This loose coupling feature means the system is more flexible, scalable, and extensible.
- ***Increased interoperability:*** Web services enhance interoperability between

software written in different programming languages, developed by different vendors, or running on different operating systems. By using the web services technology, full language and platform independence can be achieved.

- ***Easy application integration:*** The high level of abstraction provided by this technology allows developers to integrate the needed functionality to their applications easily and rapidly.
- ***Reduced complexity by encapsulation:*** All components in web services are services. So application designers do not need to worry about implementation details of the web services they are invoking. Instead, they only need to focus on the service behaviors. This will greatly reduce the system complexity and make the existing web services more reusable.

2.2 Web Service Protocols

The web service technology is designed to overcome the challenge of automating business process interactions. The key is the use of a stack of standardized protocols such as SOAP, WSDL, and UDDI. These protocols are layered and shown in Figure 2.2. The whole web services architecture resides on the top of Internet architecture. Starting from the bottommost layer, transport, this section will give a short description for each layer and the related standards used.

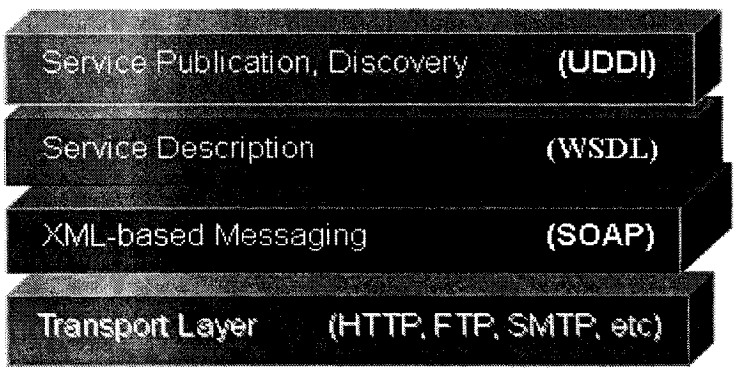

Figure 2.2 Web service protocol stack

2.2.1 Transport Layer

The transport layer is the foundation of the web services stack. It is used to do the basic

communication, addressing, and routing tasks. A good number of Internet protocols can be supported by this stack such as HTTP, SMTP (for electronic mail), FTP (for file transfer), and so on.

2.2.2 XML-based Messaging -- SOAP

XML-based messaging represents the use of XML as the basis for packing the data that is moved across applications. According to [31], there are several alternatives for XML messaging. Nowadays, the most popular and well accepted one for web service is Simple Object Access Protocol (SOAP). Alternatively, XML Remote Procedure Calls (XML-RPC) or just HTTP GET/POST can be used to pass arbitrary XML documents.

Simple Object Access Protocol (SOAP)

SOAP is an XML-based communication protocol intended for exchanging structured information in a decentralized, distributed environment [34]. SOAP is entirely written in XML, so it is independent of the programming language and operating system. SOAP can be used singly or in combination with any transport layer protocols, as shown in Figure 2.2.

SOAP messages can be categorized as request messages and response messages. Request messages allow service requesters to request a remote procedure, and response messages allow service providers to respond to such request. All SOAP messages support the '*publish*', '*bind*', and '*find*' operations in the web services architecture.

SOAP consists of three parts: an envelope to describe the content of a message, a set of data encoding rules to define a serialization mechanism, and a convention for providing remote procedure calls and responses. The basic building blocks of a SOAP message are listed as following [34]:

- SOAP Envelope: The SOAP envelope element is the root element of any SOAP message and it is mandatory. It consists of an optional SOAP header and a mandatory SOAP body.

- SOAP Header: The SOAP header element may not be present in a SOAP message. If present, it can provide a mechanism for authentication, transaction management, state information management, payment etc depending on the application requirements.
- SOAP Body: The SOAP body element must be presented in each SOAP message. The body element contains the information for objects to be processed.
- SOAP Fault: The SOAP fault element is an optional element within a SOAP body. If present, it can be used to carry error or status information within a SOAP message.
- SOAP Attachment: The SOAP attachment part is also optional. It can be used to transmit non-XML data like images, compression files, etc along with the SOAP message.

Except of language and platform independence, SOAP has the benefits of simplicity and extensibility. More important, SOAP overcomes firewalls when used together with HTTP protocol.

2.2.3 Service Description – WSDL

The public interface to web services is described by Web Services Description Language (WSDL). It is an XML-based specification for describing and locating network services. A WSDL document describes where a web service is deployed to, what operations that web service provide, and how the service requesters can invoke the web service.

To invoke a web service, a consuming application must know the service's interface, including how to structure content and which transport protocol to use. WSDL explicitly describes this interface in a standardized, machine-readable format fit for consumption by tools [41].

The WSDL specification contains two parts. One is the public interface part that describes the abstract type interface and its protocol binding. The other one is the implementation part that describes the service access information. The six basic XML

elements of a WSDL are listed below [35].

- Type: The type element contains the definitions of the data types used within the WSDL document message. The data types use the XML schema so that they can be independent.
- Message: The Message element is an abstract description of the data being exchanged. A message represents one interaction between service requester and service provider, whether it is a single message request or a single message response.
- Port Type: The port type element describes the abstract operations supported by one or more endpoints. Each operation defines an input and an output message. The port type ties a particular request and response message to a particular service.
- Binding: The binding element specifies a concrete protocol and data format for a particular port type. The binding information contains the protocol name, the invocation style, a service ID, and the encoding for each operation. There can be many bindings in a WSDL document.
- Port: The port element identifies the end points of the web service. A single end point combines a binding and a network address together.
- Service: The service element is used to define the name of the service and group all the related port elements inside a WSDL document. It defines the URL address for invoking the specified service.

The first four elements together describe the service interface, which is typically used at development time. The service and port elements describe the service location information that may be needed at either development or execution time to invoke a particular service. With the formally described interface and implementation, we can find and bind web services automatically.

2.2.4 Service Publication and Discovery – UDDI

Once a web service has been described in WSDL, it can be made publicly available. This procedure is called service publication. Service discovery is the process of locating, or finding one or more related documents that describe a particular web service using

WSDL.

UDDI [36] is the short for Universal Description, Discovery, and Integration. It is an XML-based registry for businesses worldwide to list them on the Internet. The ultimate goal of UDDI is to streamline online transactions by enabling companies to find one another on the web and make their systems interoperable. With the UDDI registry, global access to web services can be easily achieved.

The discovery of a service relies on the way it is published. In direct publication mode, the service provider sends a WSDL document directly to a service requester. Then the service consumer can retrieve the WSDL document from a local file. This is called static discovery. In indirect service publication mode, the service provider sends a WSDL document to a public UDDI registry. After that, the service consumer can retrieve the web service at run time through this registry. This is called dynamic discovery.

UDDI defines a common means to publish information about businesses and services. The information registered with the registry includes the name of the service, its description, interface, and so on. Each web service that is published with a registry server is assigned a unique key so that services with the same name can be registered at one registry server. The UDDI specification consists of a data model and a description of the UDDI API. The data model describes the following four core types of information.

- Business Entity: It provides the actual business information about a service provider such as business name, description, contact information, and category.
- Business Service: It includes non-technical information about a single web service or a group of related web services such as service name and description. A business service is a descriptive container used to group a set of web services related to a business process.
- Binding Template: This element contains technical information about service description, for example, how and where to access a specific web service. This information can be used to find and invoke a web service.
- tModel: tModel stands for technical model. A tModel is a metadata describing

type specifications as well as categorization information.

As we can see, with the layered protocol stack, the web service integration and interoperability problem has been simplified. XML solves the problem of cross-platform data encoding and formatting. SOAP defines a simple way to package information for exchange between applications. WSDL describes the details of web service, while UDDI provides a way to publish and/or discover a specific web service universally.

2.3 Web Services Architecture

The World Wide Web Consortium (W3C) [37] has introduced the web services architecture. Web services reflect a new service-oriented architecture approach. All components in a web service system are services. Some of the fundamental concepts in web services are encapsulation, message passing, dynamic binding, and service description and querying.

The web services architecture defines the following three roles [39]:

- Service Provider: A service provider creates a web service, defines a web service description, and publishes it to a service registry.
- Service Requester: A service requester is the one who consumes web services. It first searches the registry to find the needed services and then writes a client application to directly bind to the web service and invoke it.
- Service Broker (Registry): This party provides a searchable repository of service description where service providers can publish their web services and service requester can find and bind to these services.

The web services architecture is based upon the interaction between these three roles. The interaction involves three basic operations: *publish*, *find*, and *bind*. Figure 2.3 is a web services model showing the operations and the roles involved. A key benefit of the web services architecture is the ability to deliver integrated and interoperable solutions.

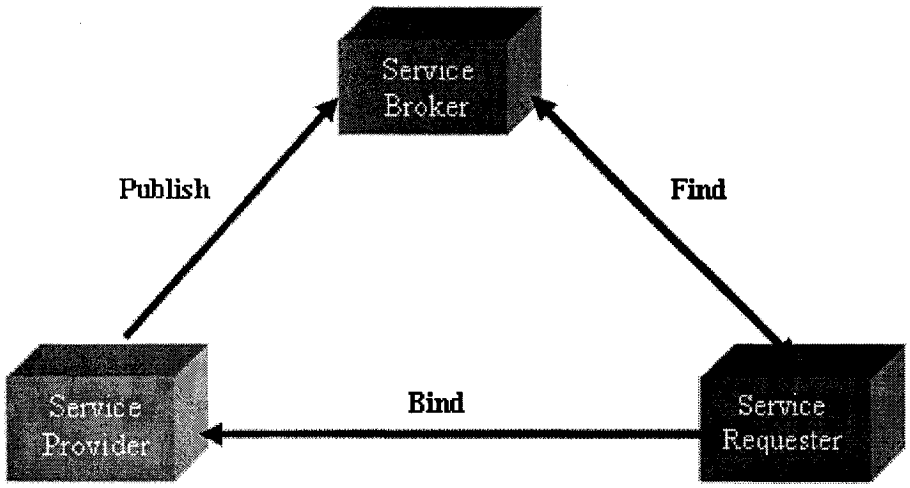

Figure 2.3 Web services publish, find, and bind model

According to Figure 2.3, the basic steps when running a web service are as following [38].

1. A service provider creates a web service and describes it using WSDL.
2. The service provider registers the web service with a UDDI registry.
3. The service requester queries the UDDI registry to locate the web service.
4. The service requester writes an application to bind to the registered web service using SOAP.
5. Finally, data and messages are exchanged freely between the service requester and provider.

The web services architecture in Figure 2.3 provides a conceptual model and a context for understanding web services and the relationships between the components of this model [42]. It is an interoperable architecture, so ensures interoperability between web services.

2.4 Web Services Composition

Based on open standards like WSDL and SOAP, web services allow any piece of software to communicate with a standardized XML messaging system [19]. The modularity and flexibility of web services make them ideal for application integration. Enterprises can easily integrate in-house business services with external web services to

provide a new web service.

The phrase 'web services composition' means the ability of one business to provide value-added services through composition of other web service components, possibly offered by different companies. The service components could be categorized as elementary web services or composite web services [7], which are defined as following.

> **Elementary Services:** *An elementary service is also called a basic service. It is an individual web accessible application that does not explicitly rely on other web services to fulfil consumer request.*
>
> **Composite Services:** *A composite service is defined as a collection of web services working together to offer a value-added service. In a recursive fashion, the new service component could in turn become a building block for an even more complex system.*

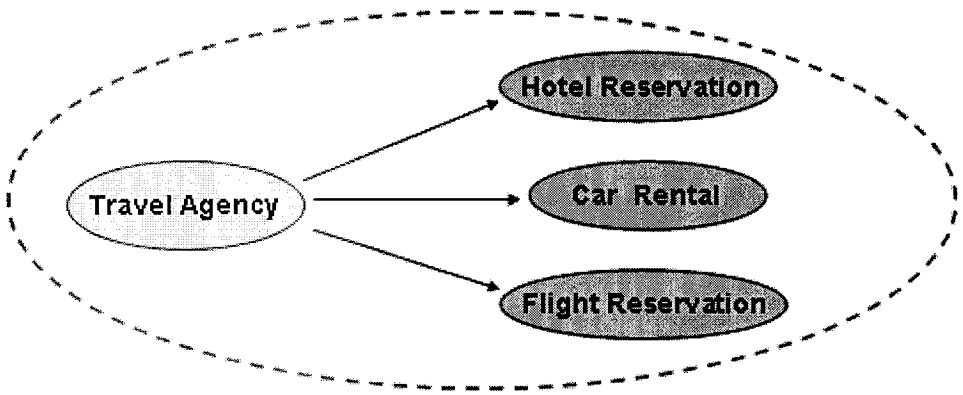

Figure 2.4 A web services composition example

Figure 2.4 shows a well-known web services composition example for a travel agency application. The travel agency web service is composed of three service components: hotel reservation component, flight reservation component, and car rental component. The travel agency doesn't need to have a priori agreements with these service providers. This allows the travel agency to have access to more web services, offering more options to its customers. The dotted oval in the diagram means that the whole travel agency service could in turn be called by other systems. In this simple example, the hotel

reservation service, flight reservation service, and car rental service are elementary services. The travel agency service is a composite service since it needs other web services to fulfill the consumer's request.

There are many benefits of composing web services together [9]. The large amount of existing services can be re-used and extended according to user's requirements. Therefore, system developers no longer need to have a complete set of service components in-house. They can easily outsource the existing service components on the web. Value-added integrated services can be provided by combining existing web services. This will make the new software development process simpler and faster.

On the other hand, web services composition is a very complex and challenging task. Several issues must be considered when integrating web services into a business process. First, due to the dynamic nature of web services, the set of web services that are capable of providing the same functionality may keep changing. Also, web services are often a part of short-term partnership. These together will make the service selection a challenging task during the composition procedure. Second, when integrating web services, not only does the requester's functional requirements need to be satisfied, but also the non-functional requirements such as cost, execution time, reliability, and so on [2]. Third, for a specific user request, there may be different ways to construct a business process. How to choose the optimal path makes the composition more complicated. At last, because of the heterogeneity of web services, there are considerable efforts in hand-coding for service composition. So how to make the composition problem dynamic, autonomous, and adaptive is quite challenging.

2.4.1 Web Services Composition Languages

Recently, several languages for web services composition have emerged from different organizations, including WSCI, BPML, BPEL4WS, XLANG, WSFL, and BPSS. The composition languages are used to express the business logic of a composite web service. The goal of these languages is to provide a process-oriented way to bind web services together. Web services composition languages build directly on top of WSDL. They use

web services described in WSDL to provide new services, which in turn are described by WSDL.

XLANG – Web Services for Business Process Design

Microsoft's XLANG is a business modeling language for BizTalk. It is a block-structured language with basic control flow structures such as sequence, switch, while, all, and pick. The goal of XLANG is to make it possible to formally specify business processes as stateful long-running interactions [26].

WSFL – Web Services Flow Language

IBM's WSFL is an XML language for the description of web services composition [27]. It is also a graph-oriented language, which relies on the concept of control links. The control flow part of WSFL is very similar to the workflow language used by MQSeries, which is also defined by IBM. WSFL Specifies two types of web services compositions. One is an executable business process known as a *flow model*, and the other one is a business collaboration known as a *global model*. WSFL is compatible with SOAP, UDDI, and WSDL.

BPEL4WS -- Business Process Execution Language for Web Services

BPEL4WS [24] is the cooperative merging of WSFL and XLANG for web services orchestration, workflow, and composition. It provides a standard XML language for expressing business processes consisting of functions defined through web services interfaces. Processes in BPEL4WS export and import functionality by using web services interfaces exclusively. BPEL4WS has both design and runtime uses. At design time, development or modeling tools can use, import, or export BPEL4WS, allowing business analysts to specify processes and developers to refine them and bind process steps to specific service implementations. The runtime choreography or workflow engine can use BPEL4WS to control the execution of processes, and invoke the services required to implement them [28].

WSCI – Web Service Choreography Interface

WSCI is another web services composition language proposed by Sun, BEA, SAP, and Intalio. It is an XML-based interface description language that describes the flow of messages exchanged by a web service participating in choreographed interactions with other services [29]. WSCI works in conjunction with WSDL or other service definition languages to describe how web service operations can be choreographed in the context of message exchanges.

BPML – Business Process Markup Language

BPML [30] is an XML-based meta-language developed by the Business Process Management Initiative (BPMI) as a means of modeling business processes. It supports both synchronous and asynchronous distributed transactions and offers reliable security mechanisms. BPML makes use of WSCI and is interoperable with BPEL4WS. BPML process can be graphically represented using the companion Business Process Modeling Notation (BPMN).

[11] gives a detailed comparison of BPEL4WS, XLANG, WSFL, BPML, and WSCI using both workflow and communication patterns. All the five languages support the basic routing constructs, i.e. sequence, parallel split, synchronization, exclusive choice, and simple merge. Emerged from XLANG and WSFL, BPEL4WS supports all the patterns that these two languages support. But BPEL4WS does not support the multi-merge pattern while BPML does.

2.4.2 Web Services Composition Approaches

Web services composition enables quick development of new applications through the reuse of existing service components. Currently, web services composition is not standardized. There are many existing proposals for web services composition. According to different criteria, web services composition techniques can be categorized as different approaches, such as manually or automated services composition [12, 15], model driven services composition [4], context based services composition [14], semantic web services composition [12, 13, 15], and so on. However, most of these web services composition techniques can be categorized into the following two types: static web services

composition, and dynamic web services composition.

Static Web Services Composition Technique

Static composition takes place during design time when the architecture and the design of the software system are planned [1]. The service components to be used are chosen and linked together statically. This approach works when business partners and service components do not change constantly. In static composition approach, application designer implements a new application manually by designing a workflow describing the interaction pattern among components [12].

As we discussed in previous section, BPEL4WS is an example of static web services composition model. The recently released Business Process Execution Language for Web Services specification is positioned to become the web services standard for composition. It allows developers to create complex processes by creating and wiring together different activities. As an executable process implementation language, the role of BPEL4WS is to define a new web service by composing a set of existing web services.

The problem of static composition is the composition procedure has to be done manually in design time. When the old web services are changed or replaced by new ones, the software architecture has to be changed and the system may need to be redesigned.

Dynamic Web Service Composition Technique

In contrast to manually describing web services composition, dynamic web services composition systems are able to adapt to the constantly changing and growing web services with minimal user intervention. The dynamic web services composition has the potential to realize flexible and adaptable applications by properly selecting and combining components based on the user request and context [12].

So far, several dynamic web services composition systems have been developed, such as eFlow, STONE, and SWORD [16]. However, most existing systems require users to request web services in strict syntax formats. [12] argued that web services composition

should be semantics-based so that a user is able to request a web service using its semantics. So far, there are lots of research activities going on in this area.

Web Ontology Language for Web Service (OWL-S) [25], formerly DAML-S, is an OWL-based web services ontology for describing the properties and capability of web services in unambiguous, computer interpretable form. OWL-S aims to solve the task of automatic web services discovery, invocation, composition, and execution monitoring. By using OWL-S, web services composition can be done dynamically.

In OWL-S, the service ontology is described by three types of knowledge, which are service profile, process model, and service grounding. The service *profile* provides a high level description of the service. The *process model* describes the execution flow of the service. The *grounding* provides a mapping between OWL-S and WSDL in order to describe how the web service has to be invoked.

In this chapter, we talked about web services protocols, architecture, and composition technique. More attention should be paid to the web services composition area because our research focuses on selecting the most appropriate web services to construct a new optimal web service.

Chapter 3

Sensor Networks Overview

Recently, advances in sensor and wireless communication make sensor networks attracting increasing research interest [45]. Sensor networks can be used in numerous areas and many new applications have started to emerge. In our research, we try to integrate web service protocols to sensor networks domain. Our focus is on the dynamic web services composition part that can be used in sensor networks. So, it is important to understand the characteristics and performance for sensor networks.

The remainder of this chapter is organized as follows. In section 3.1, we give a brief introduction about sensor networks. The sensor networks protocol stacks are presented in section 3.2. In section 3.3, we describe the main characteristics of sensor networks. Finally, we give some typical sensor networks applications in section 3.4

3.1 Sensor Network Introduction

Sensor networks are wired or wireless networks that are composed of a large number of disposable devices with sensors embedded in them. These devices, referred to as sensor nodes, are used to collect and disseminate environmental data such as light intensity, sound, and temperature. Each sensor node can consist of one or more sensors, which can be used to monitor and control the physical environment from either inside the phenomenon or remote locations.

A sensor node combines the abilities to sense, compute, and communicate. So, each sensor node in a sensor network consists of at least four key components: a sensing unit, a processing unit, a communication unit (i.e. transceiver), and a power unit. As shown in Figure 3.1, a sensor node may also have additional application-dependent components, such as location finding component, power generator, and so on [44].

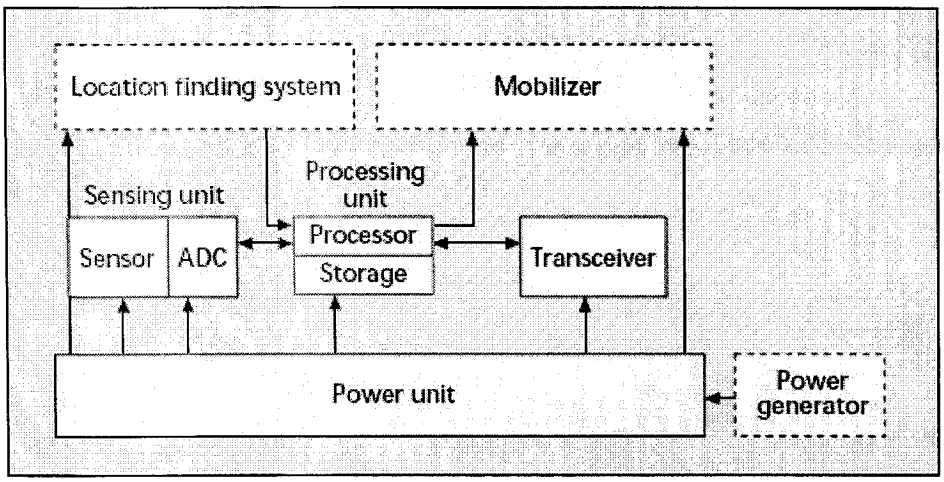

Figure 3.1 The components of a sensor node [44]

The sensor nodes have the characteristics of low-cost, low-power, multifunctional, and small size. Due to the small size and low cost characteristics, each sensor node has only limited power resources, limited processing capabilities, and limited memory. Therefore, efficient processing and communication mechanism will be needed.

The sensor nodes are usually densely deployed in a sensor field as shown in Figure 3.2. Each of these scattered sensor nodes has the capabilities of collecting and routing data. The sensing data are routed back to the sink node through a multi-hop architecture. The sink node is responsible for communicating with the task manager node through Internet or satellite.

The main functionality of a sensor network is to monitor a region of interest and detect interesting events in the network. Multiple sensors interact with each other to ensure robust, accurate detection and measurement of physical phenomena. With distributed processing, sensor networks can improve sensing accuracy by aggregating the vast amount of sensing information to provide a rich and better view of the environment.

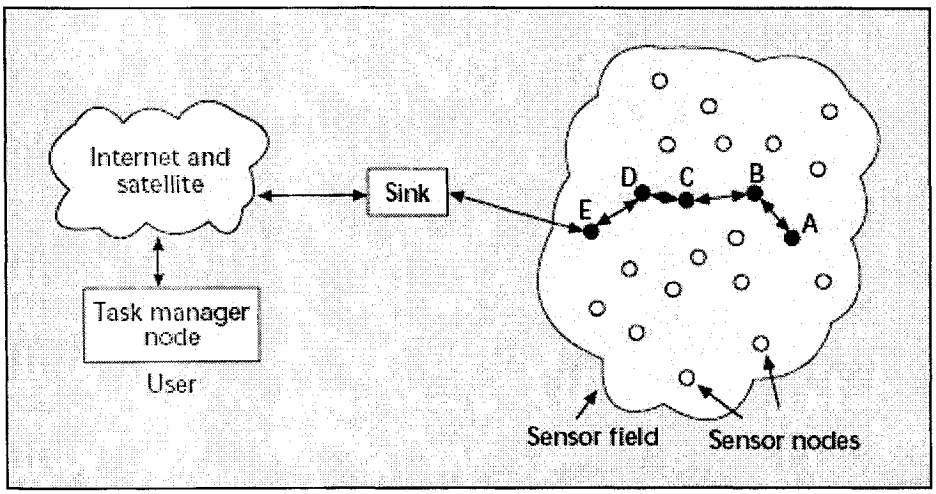

Figure 3.2 Sensor nodes scattered in a sensor field [44]

The sensor networks can be used for a wide variety of application areas. Examples include environment monitoring, traffic analysis, military surveillance, health systems, and so on.

3.2 Sensor Network Protocol Stack

According to [44], the protocol stack used by sensor nodes consists of five layers and three management planes, which are the physical layer, data link layer, network layer, transport layer, application layer, power management plane, mobility management plane, and task management plane, see Figure 3.3. These components together can provide power and routing efficiently communication between sensor nodes.

In the following, we briefly talk about the main functionality for each layer and management plane. We also give short description about the protocols used in data link layer, network layer, and application layer.

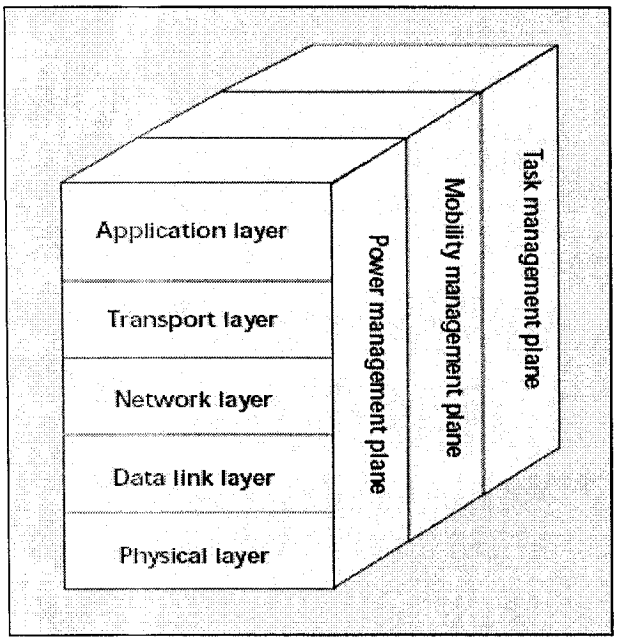

Figure 3.3 The sensor network protocol stack [44]

3.2.1 Physical layer

The physical layer is responsible for carrier frequency generation, signal detection, modulation, transmission and receiving techniques. Thus far, the 915MHz industrial, scientific, and medical (ISM) band has been widely used for sensor network. Since long distance wireless communication is expensive in terms of both energy and implementation complexity, multi-hop communication schema is widely used for sensor network. Also, multi-hop communication can effectively overcome shadowing and path loss effects.

3.2.2 Data link layer

The data link layer is responsible for the multiplexing of data streams, data frame detection, medium access control, and error control. The MAC protocol must first create the network infrastructure, establish communication links for data transfer, and must be power-aware. At the same time, the MAC protocol has to fairly and efficiently share communication resources between sensor nodes.

The Self-Organizing Medium Access Control for Sensor Networks (SMACS) protocol

[46] enables nodes to discover their neighbors and establish transmission communication without the need of master nodes. The Eaves-drop-And-Register (EAR) algorithm offers continuous service to mobile nodes.

[47] presented a Carrier Sense Multiple Access (CSMA) based MAC scheme for sensor networks. This MAC protocol supports variable but highly correlated periodic traffic in sensor networks. An adaptive transmission rate control (ARC) achieves medium access fairness.

3.2.3 Network layer

The network layer takes care of routing the data supplied by the transport layer. Sensor networks are mostly data-centric. The sensor nodes have either attribute-based addressing or location-based addressing.

The routing techniques used in network layer must be power efficient. Some data aggregation techniques may be needed to solve the implosion and overlap problems in data-centric routing. Data aggregation can be used to combine the data that come from many sensor nodes into a set of meaningful information.

The small minimum energy communication network (SMECN) [48] protocol computes an energy-efficient subnetwork of the whole communication networks. Flooding is an old distributed technique that has the properties of easy maintenance and simple route discovery. However, it has the problems of implosion, overlap, and resource blindness. Gossiping is a derivation of flooding in that it randomly selects one of its neighbors to send the data. Although it avoids the problems of implosion, the message propagation time is very long in some case.

The sensor protocols for information via negotiation (SPIN) [49] adaptive protocol are widely used in sensor network to address the deficiencies of classic flooding. SPIN uses negotiation and resource adaptation techniques. It has three types of messages: ADV, REQ, and DATA. Directed Diffusion [50] is another technique for data dissemination.

First the sink node sends out interest, i.e. a task description. As the interest is propagated throughout the sensor network, the gradients from the source back to the sink are set up. When the source has data for the interest, it sends the data along the gradient path. Low-Energy Adaptive Clustering Hierarchy (LEACH) is a clustering-based protocol. The purpose of LEACH is to minimize energy dissipation in sensor networks. The Sequential Assignment Routing (SAR) [51] algorithm creates multiple trees. This allows a sensor node to choose a tree to relay its information back to the sink.

3.2.4 Transport layer

The transport layer helps to maintain the flow of data. The TCP layer is needed to make sensor networks interact with other networks such as the Internet. In this case, communication between the user and the sink node is by UDP or TCP via the Internet. A special transport protocol is needed for the communication between the sink node and sensor nodes, or maybe a purely UDP-type protocol.

3.2.5 Application layer

Different types of application software can be built on application layer based on the sensing tasks. Nowadays, accessing sensor networks through Internet is becoming more and more important. Therefore, some application layer protocols are needed for task management and data dissemination.

Sensor Management Protocol (SMP)

System administrators manage and control the sensor network using SMP. Since the nodes in sensor network do not have global identification, attribute-based naming and location-based addressing will be used in SMP.

Task Assignment and Data Advertisement Protocol (TADAP)

TADAP deals with the issue of interest dissemination, such as the user sends their interest to the sensor networks, or the sensor nodes advertise the available data to the users, and so on.

Sensor Query and Data Dissemination Protocol (SQDDP)

SQDDP lets users to issue queries like 'temperature read by the nodes in region A', and collects incoming replies. Sensor query and tasking language (SQTL) [52] is proposed to provide large set of services.

All these protocols are open research issues, and different types of protocols may be defined for various applications.

3.2.6 Power Management Plane

The power management plane manages how a sensor node uses its power. For example, when the power level of the sensor node is low, the sensor node broadcasts to its neighbors that it is low in power and cannot participate in routing messages. The remaining power is reserved for sensing.

3.2.7 Mobility Management Plane

The mobility management plane detects and registers the movement of sensor nodes, so a route back to the user is always maintained, and the sensor nodes can keep track of who their neighbor sensor nodes are.

3.2.8 Task Management Plane

The task management plane balances and schedules the sensing tasks given to a specific region. Not all sensor nodes in that region are required to perform the sensing task at the same time. These management planes are needed so that sensor nodes can work together in a power efficient way.

3.3 Sensor Network Characteristics

Sensor networks have their own characteristics, which greatly influence the design of the sensor networks. In our project, we do not focus on sensor network design. However, understanding these characteristics will help us a lot in designing our own ontology and web service selection algorithms. Some main characteristics of sensor networks are listed below.

- Sensor nodes should be very small, cheap, and robust to environmental influences. This is because of the fact that sensor nodes are densely deployed and prone to failure in most of the cases.

- Sensor nodes have limited resources

Due to the small size of sensor nodes, the onboard resources of a sensor network – energy, communication bandwidth, computing and processing power, memory – are limited. To guarantee long-living despite the limited resources, energy efficiency must be optimized for both hardware and software components. Since communication dominates processing in energy consumption, so communication should be optimized to consume as less energy as possible. At the same time, each sensor node should process locally as much information as possible in order to minimize the total number of bits transmitted.

- Sensor nodes are densely deployed

In a sensor network, many thousands of sensors may have to be deployed for a given task. Therefore, sensor nodes need to cooperate effectively. Since multi-hop communication is expected to consume less power than the traditional single hop communication, sensor nodes use neighboring nodes to relay data in most cases.

- Sensor networks are highly dynamic, fault tolerant

On one hand, sensor nodes are prone to failure. They may die due to depleted batteries or harmful environmental influences. On the other hand, new nodes are added to replace failed ones, some nodes may move due to environmental factors like wind. Therefore, the topology of a sensor network changes quite frequently. Despite these dynamics, sensor networks should perform their tasks in a robust way. As a result, sensor networks must be tolerant to failure. Clearly, algorithms for sensor networks must be distributed or decentralized in order to prevent single point of failure.

- Sensor networks should be self-configuring and self-organizing

Since the position of sensor nodes need not be engineered or predetermined and in some

cases sensor nodes can be randomly deployed, the sensor networks protocols and algorithms should posses self-organizing capabilities. By self-organization, we mean the process of autonomous formation of connectivity, addressing, and routing structures. This process includes neighbor discovery, cluster formation, address assignment, and routing. The self-organizing characteristic also makes a sensor network more scalable.

3.4 Sensor Networks Application

Since sensor networks can provide end users with intelligence and a better understanding of the environments, the sensor networks can be used for a wide variety of application areas. Nowadays, the application areas cover at least the industrial, scientific, medical, and military domains. Examples of these applications include inventory control and factory automation [53], environment and habitat monitoring [55], health system [54], military surveillance [53], and so on.

Chapter 4

Proposed Service Selection Algorithms for Web Service Composition

Sensor networks have been experiencing rapid growth during the past few years, given the recent advances in wireless communication, digital electronics, and miniaturization design. At the same time, web services have become a promising technology because of their self-contained, loosely coupled, interoperable, and platform and language independency features.

Integrating web services technology to sensor networks domain offers many advantages. For example, the cost could be reduced significantly by leveraging the existing Internet protocols instead of proprietary protocols; 3rd party can deliver value-added web services to end users easily; by adopting web services and Internet, one can acquire and manage sensory data from all around the world. By using web services protocols, the sensor networks can be designed for more general applications, having loosely coupled data communication features.

The rest of this chapter is organized as the following. In section 4.1, we give a brief description about our Instantel project. After that, the web services composition procedure is discussed in section 4.2. Since we need to distinguish web services using non-functionality parameters, the QoS parameters for web services are introduced in section 4.3. Based on the QoS parameters, we give the *cost function* concept in section 4.4. Then, in section 4.5 and section 4.6 we present the proposed service selection algorithms for elementary services and composite services respectively. Finally, we discuss some related work in section 4.7.

4.1 Project Description

Our research project aims to bring web services into the design of sensor networks. A typical network configuration scenario is shown in Figure 4.1. In this scenario, several wireless vibration sensors and Global Positioning System (GPS) sensors are placed around a site. The vibration sensors are used to collect the vibration information for blasting applications. The GPS sensors, which are attached to the vibration sensors, are used to get location information. The relationship between the vibration sensor and GPS sensor are independent and correlated in a later stage. The vibration sensors transmit their data to a Network Capable Application Process (NCAP) that aggregates the vibration sensor data and offers a web service interface to end users. The GPS sensors transmit data to a 3rd party location provider that also offers a web service interface allowing end user to access the location data. Another 3rd party service provider integrates vibration sensor data with location information to offer visualization and analysis (V&A) functionality. As shown in Figure 4.1, there may have other web services provided by 3rd party, such as the map web service, the directory web service, and so on.

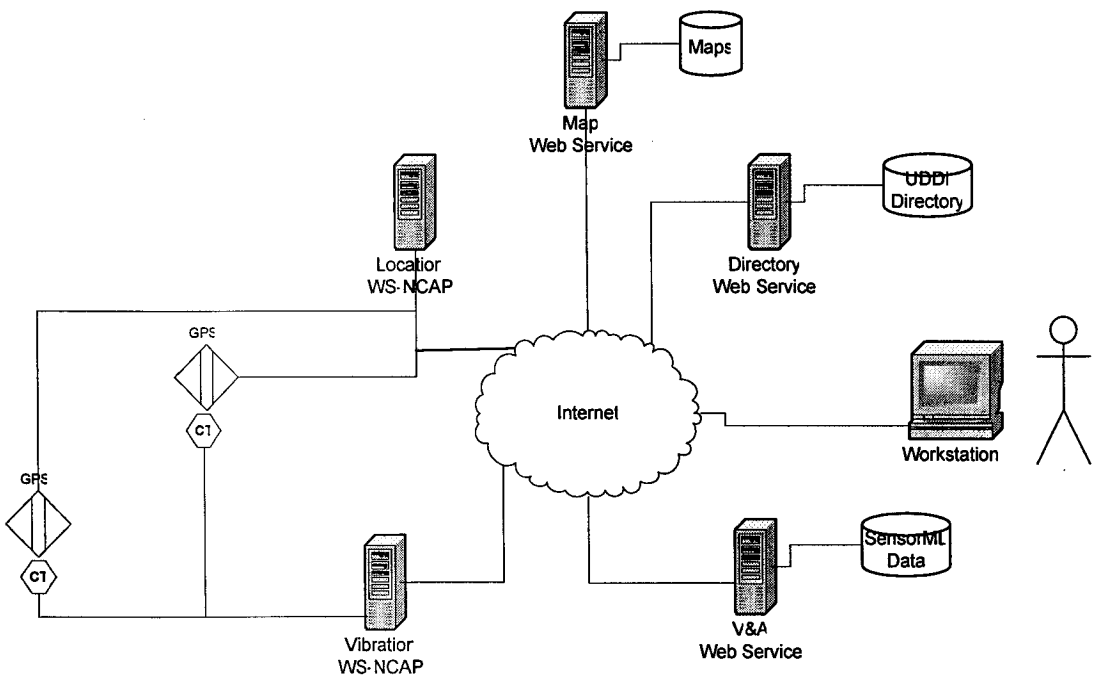

Figure 4.1 Proposed web service network configurations in sensor network

My research focuses mainly on the area of web services composition, i.e. after the sensor data has been transmitted to NCAP and a web service interface has been provided to end users. As we stated previously, value-added services can be provided by combining existing web services together. However, how to choose the appropriate web services to combine and optimize the composite service is a challenging research issue, especially in sensor networks area where the system resources are limited.

In this thesis, we present a novel web services selection approach that can be used in web services composition process. The proposed mechanism can be used in sensor networks as well as other application domains. In the proposed service selection algorithm, user preferences and some non-functional parameters of web services, like *price* and *execution time*, are used to be the selection criteria.

Due to the fact of the limited storage, computation, processing, and energy resources in sensor networks, the proposed mechanism must be fast, efficient, and resources effective. The proposed service selection algorithm must also be dynamic and fault tolerant because of the dynamics and fault tolerance features of sensor networks.

4.2 Web Services Composition Procedure

Before discussing the web services composition procedure, we first give the definition of service community [7].

> **Service Community:** *A service community is a group of web services that provide the same functionality, while their non-functionality (such as execution time, delay, cost, etc) may be different.*

For example, many web services provide the functionality of translating English to French. Even though they all can do the translation properly, their response time, reliability, and price may be different. In web services composition, we need to choose the most appropriate web service according to their non-functional parameters and user's preferences.

Web services composition is a very complicated process, which includes the following four steps: service discovery, service selection, service binding, and service execution, shown in Figure 4.2.

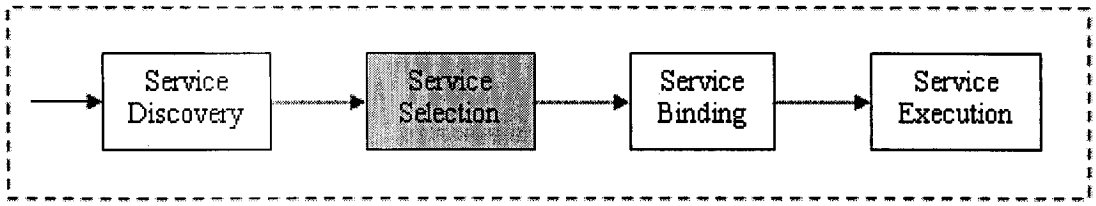

Figure 4.2 Web services composition diagram

Service discovery is the procedure of finding the suitable web services that satisfy the requester's functional requirements from the service community. This can be done by searching the UDDI registry to find the needed web services. The UDDI registry contains the information for describing and advertising the web services. The UDDI registry may also contains some non-functional parameters that can be used to find the best web service, as proposed in [15]. Some semantic information may also be included in UDDI registry, which will greatly benefit the procedure of service discovery.

After service discovery, a large amount of web services, which perform the similar functionality, will be found. The service selection step aims to choose the appropriate web services (atomic and/or composite) for composition. The goal of service selection is to make sure that the combined service fulfills not only the requester's functional requirements, but also the non-functional requirements. Moreover, service selection step should guarantee that the combined service has the optimal performance, in both functional and non-functional respects.

Service binding is responsible for combining the selected web services together either manually or automatically. Finally, the integrated web service will be executed and monitored.

In order to support dynamic web services composition, two problems have to be addressed. One is web services discovery and selection. The other one is adapting to web service changes and failure. The main motivation of our research is to make intelligent web services selection decision for business process requester. To differentiate the set of web services that provide the similar functionality, the *cost function* concept is introduced based on the web services Quality of Service (QoS) parameters such as cost, execution time, reliability, security, and so on.

4.3 QoS Parameters for Web Services

In order to select the appropriate web services for composition, we must first find matches between the advertised web services and end users' functional requirements. Usually, many component services are able to execute a given task, each with different levels of pricing and quality. Therefore, an effective mechanism is needed to distinguish these web services. The non-functional characteristics – Quality of Service (QoS) parameters – act as crucial criteria for web services selection procedure.

In our research, we propose two web service selection algorithms based on the QoS parameters and users' preferences. One algorithm is used to select the best elementary service. The other one is designed to choose the optimal path for the composite service. The aim of these service selection algorithms is to optimally select component services at run time according to users' preference and non-functional parameters like cost, execution time, and so on.

According to [8, 10, 18], some generic Quality of Service parameters for web services are listed and defined below.

- **Price** – It is the amount of money that a service requester has to pay for executing the operation of a web service.
- **Execution time** – It measures the expected delay a web service to process its sequence of activities.

- **Reliability** – The reliability of a web service is the probability that a request is correctly responded within the expected time frame. It is the overall measurement of a web service to maintain its service quality.
- **Availability** – The availability of a web service is the probability that the service is accessible.
- **Latency** – Latency measures the network delay. It is the round-trip time between sending a request and receiving the response of the web service.
- **Throughput** – It measures the maximum requests that a web service can accept at the same time.
- **Reputation** – The reputation of a web service is a measurement of its trustworthiness. This parameter mainly depends on end user's experiences of using the web service.
- **Security** -- Security for web services means providing non-repudiation and confidentiality by authorizing the parties involved, encrypting messages, and providing access control.

For our research purpose, we only consider five non-functional parameters, which are execution price, execution time, reliability, latency, and throughput. But this can be easily extended to include other QoS parameters. For each parameter, users can give their preference according to the request requirements. These five parameters, together with their corresponding preferences, are designed to be the selection criteria for our services selection algorithms.

In order to get these quality-based parameters, several different approaches may be applied. Some parameters can be published into the UDDI registry when the web services are registered. This can be done by using either syntactic or semantic format. For example, the service providers can publish the *price* of their services when they register their services into service broker. Some parameters can be computed at run time, a good example for that is *latency*. Some parameters, such as *reputation* and *reliability*, can be acquired from a 3rd party broker according to users' feedback [6].

The methods of how to get these parameters will not be discussed in our thesis. Based on the QoS-based web service parameters, our research only focuses on finding the proper web services to optimize the composition. Some web services quality models that are similar to ours are proposed in [4, 7, 8].

4.4 Cost Function Definition

In order to find the best web service among a service community, the *cost function* concept is introduced.

> **Cost Function:** *A cost function is a weighted function that represents the total costs of invoking an elementary or composite web service.*

The value of cost function is calculated based on the web service's QoS parameters and the service requester's preferences. As mentioned in previous section, in our research, only five non-functional parameters are considered, which are *throughput, latency, execution time, reliability*, and *price*.

The cost function calculation for a specific web service among a service community can be divided into the following three steps.

Step One: Filtering Phase

After the user provides the selection criteria for QoS parameters and their corresponding preferences, the first thing we need to do is to check each web service to see if it satisfies the given QoS criteria. If not, there is no need to compute its cost function and we simply delete this web service from its service community. Otherwise, we keep this web service. After all the web services are checked, we simply go to step two.

Step Two: Averaging and Scaling Phase

For computing the cost function of a web service, the absolute value of the QoS parameters does not make sense. So, we want to use the relative value. For each

service community, we first calculate the average value of each QoS parameter. After that, the relative value of each parameter can be computed by dividing the actual value with the corresponding average value. For example, the relative value of a web service's *price* parameter can be calculated by the following equations:

$$Price_{av} = \left(\sum_1^N price\right) / N \quad (1)$$

$$Price_{relative} = price \,/\, Price_{av} \quad (2)$$

where

- N is the total number of qualified web services in the service community

Step Three: Weighted Phase

In this step, the user's preferences for each parameter will be considered. There are two situations. Some of the parameters could be negative, i.e. the higher the value, the lower the quality. Examples include price, latency, and execution time. Other criteria are positive, i.e. the higher the value, the better the quality. This includes criteria such as throughput and reliability. For negative criteria, values are weighted according to formula (3). For positive criteria, values are weighted according to formula (4).

$$V_i = W_i * N_{relative} \quad (3)$$

$$V_j = W_j \,/\, P_{relative} \quad (4)$$

Where

- $N_{relative}$ and $P_{relative}$ represent the relative value for Negative and Positive parameter respectively
- W_i, W_j represent the user preferences

Combining equation (1) to (4), the cost function of a web service S is defined as the following.

$$F(s) = w_1/(thru/thru_{av}) + w_2*(late/late_{av}) + w_3*(exec/exec_{av}) + w_4/(reli/reli_{av}) + w_5*(price/price_{av}) \quad (5)$$

where

- thru, late, exec, reli, and price correspond to the QoS parameters throughput,

latency, execution time, reliability, and price.

- the subscript $_{av}$ means the average value of the service community which provide the same functionality.
- the value w_i corresponds to the user preference for each parameter.

In summary, by using this definition, the smaller the cost function value is, the better the web service would be. So, the goal of our service selection algorithm is to find the web service that has the minimal cost function value. The question of finding the best elementary service among a service community is very simple, because the cost function of each web service can be easily calculated by using equation (5). However, to find the best composite service is quite challenging. These two algorithms are discussed in detail in the next two sections.

Please note: in the following of this thesis, when we say selecting a web service from a service community, we always means select the one with minimum cost function. When we say selecting two web services from a service community, it means we select the two service components whose cost functions have the minimal and the second minimal value, and so forth.

4.5 Selection Algorithm for Elementary Services

The idea of selecting the best elementary service from a service community is straightforward. After computing the cost functions for all the web services, we can sort them in increasing order according to their cost functions. Then the first element in the list will be the best elementary web service.

Here is the algorithm for calculating the cost function and selecting the best elementary service among a service community.

Input:

- A list of elementary web services S_1, S_2, ... Sn that perform the same functionality

- The QoS parameters (*throughput Pt, latency P_l, execution time Pe, reliability Pr,* and *price Pp*) for each elementary web service.
- The QoS parameters *throughput, latency, execution time, reliability,* and *price* entered by service requester
- The preference parameters w_1, w_2, w_3, w_4, and w_5 that correspond to *throughput, latency, execution time, reliability,* and *price* respectively.

Output:

- The best elementary web service satisfying all the requester's requirements

Algorithm 4-1: selecting the best elementary service

```
//step 0: initialization and read the QoS and preference input from end user
construct a new list L1 and read all the web services
read QoS and preference input

// step 1: delete all the services in list L1 that do not satisfy the requirements
for (i=0; i<n; i++) {
    if ((throughput >= Pt & latency <= Pl & executionTime<=Pe
        & reliability>=Pr & cost<=Pc) == false) then
        delete Si from list L1
}

// step 2: calculate the average value of each parameter
for each service Si in list L1 {
   for each weight coefficient wi \= 0 {
        add the corresponding parameter to related sum
   }
}
calculate the average value for each of the five QoS parameters

// step 3: calculate the cost function for each service
```

for each service Si in list L_1

calculate the cost function $f= w_1/(t/t_{av}) + w_2*(l/l_{av}) + w_3*(e/e_{av}) +w_4*(r/r_{av}) + w_5*(c/c_{av})$

// step 4: get the best elementary service

sort list L_1 in increasing order according to cost function

select the head element from list L_1, which has the minimal cost function

return the selected head element

In the above algorithm, we calculate the cost function for each web service in step 1, 2, and 3. Step 1, 2, 3 correspond to the filtering phase, averaging phase, and weighted phase respectively. In step 4, we sort all the web services in increasing order. This can be done by using the fast sort algorithm, which has the complexity of O(*NlogN*).

4.6 Selection Algorithm for Composite Services

As we can see, choosing the best elementary web service is quite obvious and simple. But in real life, we need to compose several web services together from time to time. The problem of finding the optimal composite service is much more complicated, because we can not simply combine the best elementary web services together. Actually, the result of composing several best elementary services together may not be optimal. In some cases, it may not even satisfy the user's requirements. The problem of finding the best composite service is similar to the optimal path problem.

In order to find the optimal composite service, theoretically we need to form all the possible paths by combining related web services together. After that, we can find the best composite service by comparing their performance. However, combining all the web services together will consume huge computation and storage resources, especially when the number of web services that provide the same functionality gets larger. In some cases, this is an impossible task due to the limited resources available. For example, in sensor network, the sensor nodes only have limited resources and limited power. Moreover, the compositional procedure of the composite service must be very fast because most of the

sensor network applications are real-time applications.

By introducing the concept of cost function, we argue that in most cases it is not necessary to combine all the web services together in order to find the optimal path. In this section, we propose a new service selection algorithm that can be used to form the optimal path of a composite service. The proposed algorithm consumes very limited resources and runs efficiently.

The idea of the proposed selection algorithm is very simple. We choose only a few (depends on the user's requirements and the available resources) web services from each service community according to their cost function. After that we compose these limited number of web services together to form composite services. The cost functions of the resulting composite services are then calculated and the optimal composite service can be easily selected. This algorithm is shown in the following.

Input:

- N, number of web services that are selected from each service community to participate the composition process
- a list of web services, each web service has a sequence number
- The QoS parameters (*throughput Pt, latency* P_l, *execution time Pe, reliability Pr,* and *price Pp*) for each elementary web service.
- The QoS parameter *throughput, latency, execution time, reliability,* and *price* entered by service requester
- The preference parameters w_1, w_2, w_3, w_4, and w_5 that correspond to *throughput, latency, execution time, reliability,* and *price* respectively.

Output:

- The best composite web service satisfying all the requester's requirements

Algorithm 4-2: selecting the best composite service among limited number of services

Step 0: initialization, read the QoS and preference input from user

Step 1: delete the web services that do not satisfy the user's requirements

Step 2: calculate the cost function of each satisfied elementary service and sort them in increasing order

Step 3: select N web services from each service community (note: only these web services will participate the composition procedure)

Step 4: form all possible composite services from this limited number of web services and delete the ones that do not satisfy the user's requirements

Step 5: calculate the cost function for each composite service

Step 6: compare the cost function of these composite services and select the optimal one.

Step 7: return the best composite service

In step 5 of this algorithm, we calculate the cost function for each composite service. In order to do that, we first need to compute the QoS parameters of the composite service. Suppose a composite service S_{com} is constructed by three web services S_1, S_2, and S_3 sequentially, the five QoS parameters of S_{com} are calculated as following.

- For parameter price, $price_{com} = price_1 + price_2 + price_3$
- For parameter delay, $delay_{com} = delay_1 + delay_2 + delay_3$
- For parameter execution time, $exec_{com} = exec_1 + exec_2 + exec_3$
- For parameter reliability, $reli_{com} = reli_1 * reli_2 * reli_3$
- For parameter throughput, $thru_{com} = \min \{thru_1, thr_2, thru_3\}$

After getting the five QoS parameters for each composite service, we can follow the same filtering phase, averaging phase, and weighted phase to calculate the cost function for each composite service.

By combining a small number of web services together, the compositional process will be very fast and only limited resources will be needed. In addition, we argue that in most of the cases we can get the same compositional result as combining all the web services together. This can be illustrated by our experiment results shown in next chapter.

The algorithm 4-3 can be used to verify the feasibility of our proposed approach. The goal of this algorithm is to check how many web services needed from each service community in order to get the optimal composite service. In this way, we can determine the number N that was used in algorithm 4-2.

In algorithm 4-3, we assign a sequence number to each service community. The sequence number represents the execution sequence in the overall composed web service. Since concurrent execution is possible, there may exist situations that two or more service communities hold the same sequence number.

The idea of algorithm 4-3 is as follows. We first calculate the cost function for each elementary service and sort them according to their cost functions. Then, we form all possible composite services that satisfy the requester's requirements, calculate their cost functions, and get the optimal solution S_k. After that, we try to find how many elementary services needed from each service community in order to get the same optimal composite service. To do so, we first select one web service from each service community, compose them together, and check if the composition result is the same as S_k. If it is, then exit. Otherwise, add one more web service from each service community at one time until we get the best composition result, namely S_k.

Input:

- A list of web services, each web service has a sequence number N
- The QoS parameters (*throughput Pt, latency P_l, execution time Pe, reliability Pr,* and *price Pp*) for each elementary web service.
- The QoS parameter *throughput, latency, execution time, reliability,* and *price* entered by service requester
- The preference parameters w_1, w_2, w_3, w_4, and w_5 that correspond to *throughput, latency, execution time, reliability,* and *price* respectively.

Output:

- The optimal composite service S_k

- N, i.e. the number of services needed from each service community for getting the optimal composite service S_k

Algorithm 4-3: determining the number of web services needed from each service community in order to get the optimal composite service

Step 0: initialization, read the QoS and preference input from user

Step 1: select all services that satisfy the requirements and put them into corresponding lists

Step 2: for each service community calculate the cost function for each web service and sort them in increasing order

Step 3: construct originList which holds atomic web services for composition, construct a resultList which will hold the composition result

Step 4: calculate all the possible composition paths that satisfy the requirements, compute their cost function and get the optimal one.

Step 5: check how many services needed in each service community, in order to get the optimal composition result

```
for (int i=0; i<N; i++) {
   compute and put all the satisfied composite service into a list and
   calculate the cost function of them and sort
   if( the cost function of the head of the list  == the cost function of the
      optimal composite service) then
        write the result and break;
}
```

Step 6: return the optimal composite service, and the number of services needed from each service community to get this best result.

From step 1 to step 4, we concentrates on composing elementary services together and computing the cost function for all the satisfied composite services. Most of these steps are similar to either algorithm 4-1 or algorithm 4-2. In this part, we actually construct all the possible composite services. The second part of the algorithm, step 5, focuses on finding how many web services needed from each service community, in order to get the

optimal composite service.

In brief, the proposed selection algorithm 4-2 will significantly reduce not only the computation and storage resources, but also the execution time. The reason for that is our approach will simplify the problem of $O(N^k)$ to O(1). The details of the complexity analysis are discussed further in chapter 6.

If you want to save the resources further and execute faster, you may apply the linear programming technique [7] or the knapsack technique [6] after selecting limited number of web services from each service community.

4.7 Related Work

So far, there are many proposals or standards for web services composition like BPEL4WS (from IBM, Microsoft, BEA, etc) and WSCI (from Sun, BEA, SAP, etc). Each player tries to get the biggest piece of cake. At the same time, the researchers have proposed a variety of techniques that can be used for web services composition. For example, the service composition problem can be viewed as a planning problem [21]; it can be accomplished by using rules and services expressed as a knowledge base and topic ontology [22]; it can also be achieved by using today's workflow management techniques [23], and so on. Therefore, it is still unclear where the industry is going with the various standards. References [1, 3, 5, 17] summarize the current solutions for web service composition.

In our research, we focus on the development of web services selection approach considering both functional and non-functional requirements. By using QoS parameters, we can easily calculate the cost function for each elementary service. Furthermore, the optimal web service can be chosen by sorting web services according to their cost function.

Some related work on QoS has been done. [10] discusses the QoS issues for web services. They give detailed explanation about availability, security, response time, throughput, etc. Reference [2, 7, 23] explains the importance of QoS parameters in web services selection

and composition.

Reference [4] introduces a model for web services discovery with QoS parameters. In this model, they first use functional requirements to discover proper web services. Then the quality of service requirements would enforce a finer search. They classify web service QoS parameters as run-time related QoS (e.g. reliability and availability), transaction support related QoS (e.g. integrity), configuration management and cost related QoS (e.g. cost and scalability), and security related QoS (e.g. authentication and authorization).

In [2], the authors propose a mechanism for computing QoS parameters. They claim they can achieve the dynamic and fair computation of QoS values of web services through a secure active users' feedback and active monitoring.

In [6], the authors propose a broker-based framework to dynamically integrate and adapt QoS-aware web services. In this framework, the QoS brokers can make intelligent service selection decisions according to functional and non-functional criteria.

The closest work to ours is [7]. In [7], the authors present a QoS-aware middleware for web services composition. They propose a quality-driven approach to select component services for a composite service. Then they use a linear programming technique to solve the service selection problem. However, this solution is too complex for run-time broker decisions.

[8] also proposes a QoS-driven middleware for web services composition. They select elementary service by optimizing them based on QoS. The global optimization for composite service is done by using global planning technique.

In [9], the authors design a service selection algorithm used by QoS brokers. They do so by model the problem as the Multiple Choice Knapsack Problem. They define 'utility function', which is similar to our cost function concept, to maximize the benefits the clients receive and minimize the cost the users pay.

Although the work [7, 8, 9] somehow is related to our research, none of them can be used in sensor networks domain because of the computational and storage limitation. Our proposed service selection algorithms solve this problem efficiently. At the same time, we take into account both functional and non-functional requirements as well as the user's preferences for each QoS parameters. The correctness of our proposed algorithms can be proved by the experimental results shown in Chapter 5. In order to deploy our proposed algorithms into sensor networks, we can use the similar idea described in [45].

In [45], a web services based architecture for sensor networks is proposed, in which WSDL is used for describing web services and SOAP is used for formatting the messages. In the proposed architecture, sensor nodes are service provider and the user applications are service consumer. One or several sensor nodes can provide a service, which is published to the sink nodes. Some web services may also be combined together to provide a high level service. The user applications can subscribe a web service from the sink nodes through the Internet.

The sink node also serves as a service broker. It caches all the published web services and maintains a routing table for these services. If a subscribed web service is available, the sink node will forward the query to the service provider. Otherwise, it will broadcast the query to all the sensor nodes, or it may combine the available web services to come up with a new service satisfying the query.

In summary, although some work discussed web services composition problem based on QoS parameters, no existing work can be used directly in sensor networks domain. Our proposed service selection algorithm only consumes limited system resources. Therefore, it can be used in the circumstances where only limited resources are available.

In this chapter, we first discussed the web services composition procedure and QoS parameters for web service. Then we introduced the cost function concept that can be used to distinguish and choose web services. After that two web service selection

algorithms were proposed, one for elementary service and the other for composite service. Finally, we discussed some related work. In next chapter, we will give the implementation result that can be used to verify the correctness and effectiveness of our proposed mechanism.

Chapter 5

Implementation and Experiment Results

We conducted a series of experiments to investigate the effectiveness of our proposed techniques. The correctness of the proposed algorithm 4-1, *selecting the best elementary service*, is obvious. Therefore, all the experiments conducted in this chapter aim to verify the effectiveness of the algorithm 4-2, *selecting the best composite service among limited number of services.* In order to do that, we run the algorithm 4-3, *determining the number of services needed from each service community in order to get the optimal composite service,* with different QoS parameters and selection criteria. The total number of experiments conducted is 5400, which is large enough to get a conclusion.

In section 5.1, we first describe the web services composition scenario used by our experiments. The system requirements and some assumption for our experiments are given in section 5.2. Finally in section 3, we list all the experiment results, which include the QoS parameters for each service community, the selection criteria from end user, and the results after running algorithm 4-3.

5.1 Simulation Design and Scenario

The two basic relationships among individual services are sequential and parallel. There are also some other relationships such as conditional and loop. According to [6], all these relationships can be converted to a sequential model. So, if our proposed algorithm works for a sequential model, it will also work for the general case. However, in our experiment scenario, we consider the composite service with both sequential and parallel execution paths to make it more interesting.

The following figure 5.1 shows a compositional scenario for language translation. Starting from Chinese, we want to translate the language into German. As can be seen from Figure 5.1, there are three possible paths to perform this task: Chinese → English →

French → German; Chinese → French → German; Chinese →English → German. The goal of this experiment is to find the best composite service among all the possible combinations.

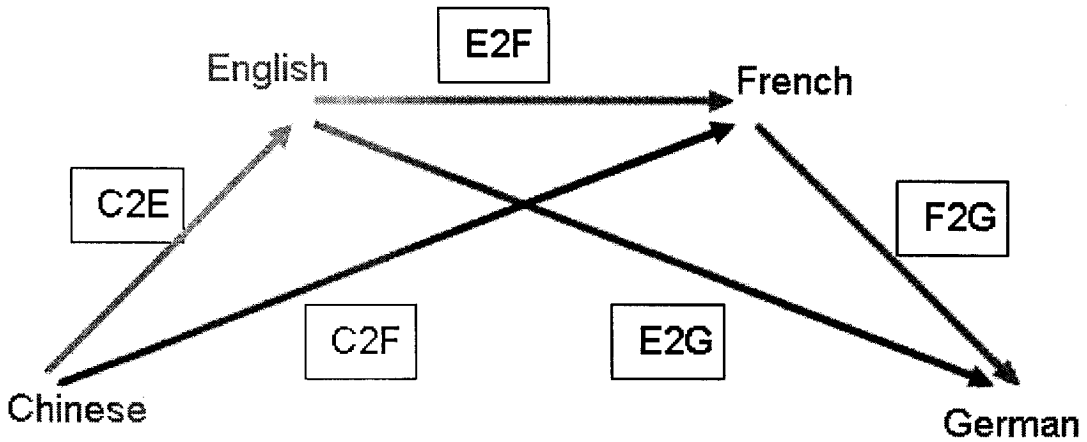

Figure 5.1 A language translation scenario

5.2 System Requirements and Assumptions

All the experiments are conducted on a Pentium 4 computer with a 2.2GHz CPU and 512MB RAM. We implemented our algorithm in a WebLogic 8.1 Workshop platform. The database server is a built-in PointBase server.

Since the number of web services in each service community is very large, so we can only simulate these web services instead of actually implement them. However, the whole system is developed as a web service, which can be called using standard protocols from all around the world.

The QoS parameters for each web service are generated randomly based on the given range. These generated values are stored in the database for later usage. In our research, we only focus on selecting the right web services to compose a value-added composite service. Therefore, the service discovery mechanism and the way of finding the QoS parameters are not our concerns. Figure 5.2 shows the user interface for getting the end user's selection criteria.

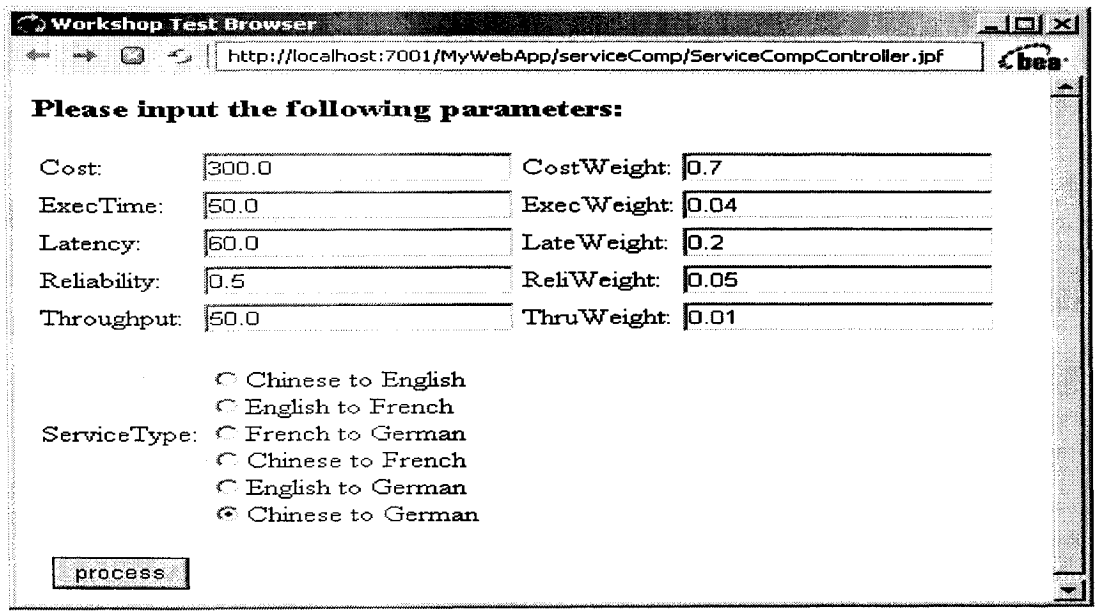

Figure 5.2 The user interface of our experiment

Based on our scenario and the given computer system, the maximum number of web services among each service community could not exceed 85. Otherwise, there will be an out of memory error. The reason for that is in our algorithm 4-3, we want to compose all the possible paths in order to find the optimal one. This needs a huge storage and computation resources. This is why we need to find a more efficient and effective service selection technique for service composition, especially in some resource and/or time limited situations such as in the area of sensor networks.

5.3 Experiment Results

This section lists all the simulation data and results. In total, we conducted four separated groups of experiments based on different QoS parameters and client requirements. The first three of them are general cases that can be used to verify the effectiveness of our proposed technique. The last experiment is a special case, in which the selection criteria only consist of one parameter.

For each experiment, the number of services that provide the same functionality ranges from minimum one to maximum eighty-five (see Table 5-3). Due to the limited memory resource we have, eighty-five services for each service community are the maximum

number we can reach based on our given scenario. For each experiment, we conducted one hundred simulations. Hence, there are totally 1800 simulations conducted for each experiment.

The experiment data are organized as following. For each experiment, we first list the QoS parameters range for each service community. Then the user selection criteria are listed. Finally, we give the simulation results and the corresponding diagram. The specific description for each table is detailed in section 5.3.1.

5.3.1 Experiment 1: *all the possible compositional paths satisfying the selection criteria*

In this experiment, the user selection criteria are very low, which makes all the possible compositional services in our scenario satisfy the user's requirements. For example, based on the given parameter ranges shown in Table 5-1, the maximum price the user needs to pay is 300 dollars (taking the path of C2E → E2F → F2G). However, the user requirement is that the price for the composed service should be less than or equal to 300 dollars. Therefore, all the composite services satisfy the user's requirements.

Table 5-1 below shows the QoS parameters range for the service communities. The specific parameters for each web service will be randomly generated based on the given range. Table 5-2 lists the end user's requirements and preferences of each parameter for the composed service.

Table 5-1: QoS parameters for each service community in experiment 1

	Throughput (Kbps)	Latency (ms)	Reliability (%)	Exec. Time (ms)	Price (dollar)
C2E	100 ~ 200	1 ~ 20	80 ~ 100	5 ~ 15	50 ~ 100
E2F	100 ~ 200	1 ~ 20	80 ~ 100	5 ~ 15	50 ~ 100
F2G	100 ~ 200	1 ~ 20	80 ~ 100	5 ~ 15	50 ~ 100
C2F	100 ~ 150	1 ~ 30	80 ~ 100	5 ~ 20	50 ~ 150
E2G	100 ~ 150	1 ~ 30	80 ~ 100	5 ~ 20	50 ~ 150

Table 5-2: user's selection criteria for the composite service in experiment 1

	Throughput (Kbps)	Latency (ms)	Reliability (%)	Exec. Time (ms)	Price (dollar)
Value	≥50	≤60	≥50	≤50	≤300
Weight	1%	20%	5%	4%	70%

The summary of the experiment results is shown in Table 5-3. In this table, the column gives the total number of web services in each service community. The row tells how many times we can get the optimal result when only one, two, three, etc, services are chosen from each service community for composition. For example, when the number of services in each service community is ten, then there are 89 out of 100 cases we can get the best compositional result when only one service is chosen from each service community to compose. If we choose one more service, i.e. two services, from each service community, then there are 99 (i.e. 89 plus 10) out of 100 cases we can get the best composite service.

Table 5-3: The summary of the experiment results for experiment 1

# of Services in community / # services chosen for composition	1	2	3	4	5	6	7	8	9	10
1	100	94	88	91	88	93	89	87	92	89
2		100	100	100	97	99	91	99	100	99
3					100	99	93	100		100
4						100	100			

Table 5-3: The summary of the experiment results for experiment 1 (cont.)

# of Services in community / # services chosen	20	30	40	50	60	70	80	85	Summary
1	91	90	84	90	87	87	85	87	**1612 (89.56%)**

2	99	100	98	100	97	98	100	100	**1783 (99.06%)**
3	100		100		100	100			**1798 (99.89%)**
4									**1800 (100%)**

From Table 5-3, we can easily see that among all the 1800 simulations, 89.56% of them can get the optimal compositional solution by choosing only one service from each service community. If we choose three web services from each service community, then we have 99.89% chance to get the best composite result. Note, when we say choosing a web service from the service community, we always choose the one that has the minimum cost function among all the available services. Figure 5.3 gives a better view about the results of experiment 1.

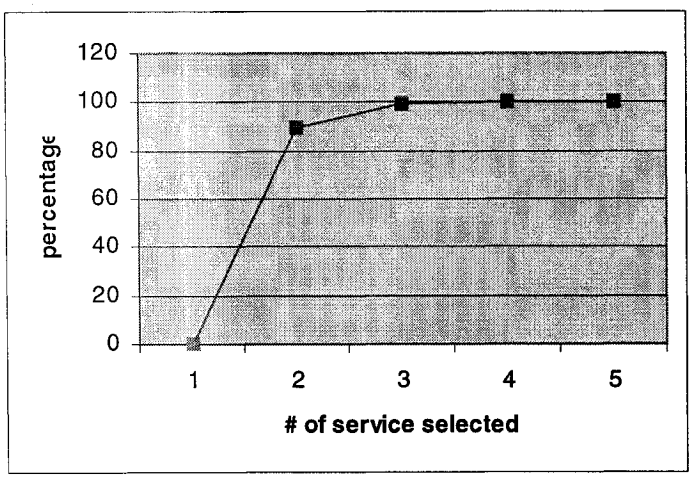

Figure 5.3 Overall simulation results of experiment 1

5.3.2 Experiment 2: *each elementary service satisfies the user requirements, but not all the compositional paths*

In this experiment, the user selection criteria are a little higher than experiment 1. In this case, we guarantee that each elementary web service satisfies the user's requirements. But, this does not necessarily means that all the composed services also satisfy the user's requirements For example, according to the selection criteria shown in Table 5-5, the price for the composed service should be less than or equal to 200 dollars. However, since the values of the QoS parameters are randomly generated, the price for some

composite services obviously exceeds that limitation (see Table 5-4).

Table 5-4 below shows the QoS parameters range for each web service community. Table 5-5 lists the end user's requirements and preferences of each parameter for the composed service. The total results of experiment 2 are summarized in Table 5-6.

Table 5-4 QoS parameters for each service community in experiment 2

	Throughput (Kbps)	Latency (ms)	Reliability (%)	Exec. Time (ms)	Price (dollar)
C2E	100 ~ 200	1 ~ 20	80 ~ 100	5 ~ 15	50 ~ 100
E2F	100 ~ 200	1 ~ 20	80 ~ 100	5 ~ 15	50 ~ 100
F2G	100 ~ 200	1 ~ 20	80 ~ 100	5 ~ 15	50 ~ 100
C2F	100 ~ 150	1 ~ 30	80 ~ 100	10 ~ 20	100 ~ 150
E2G	100 ~ 150	1 ~ 30	80 ~ 100	10 ~ 20	100 ~ 150

Table 5-5: user's selection criteria for the composite service in experiment 2

	Throughput (Kbps)	Latency (ms)	Reliability (%)	Exec. Time (ms)	Price (dollar)
Value	≥100	≤30	≥50	≤25	≤200
Weight	1%	20%	5%	4%	70%

Table 5-6: The summary of the experiment results for experiment 2

# of Services in community / # services chosen for composition	1	2	3	4	5	6	7	8	9	10
0	70	19	4	1	0	0	0	0	0	0
1	100	75	69	70	80	74	69	74	71	74
2		100	97	96	96	96	89	96	92	92
3			100	99	99	100	98	99	96	98
4				100	100		100	100	97	100

5									98	
6									100	

Table 5-6: The summary of the experiment results for experiment 2 (cont.)

# of Services in community / # services chosen	20	30	40	50	60	70	80	85	Summary
0	0	0	0	0	0	0	0	0	**94 (5.22%)**
1	81	79	77	86	73	77	86	78	**1393 (77.39%)**
2	97	93	96	94	93	96	97	96	**1716(95.33%)**
3	99	97	100	99	100	99	98	99	**1780 (98.89%)**
4	100	99		100		99	100	100	**1795(99.72%)**
5		99				99			**1796 (99.78%)**
6		99				100			**1799(99.94%)**
7		99							**1799(99.94%)**
8		100							**1800 (100%)**

From Table 5-6, we can see that by selecting three web services from each service community, we have 98.89% chance to get the optimal composition solution. This result can be seen from Figure 5.4 more intuitively.

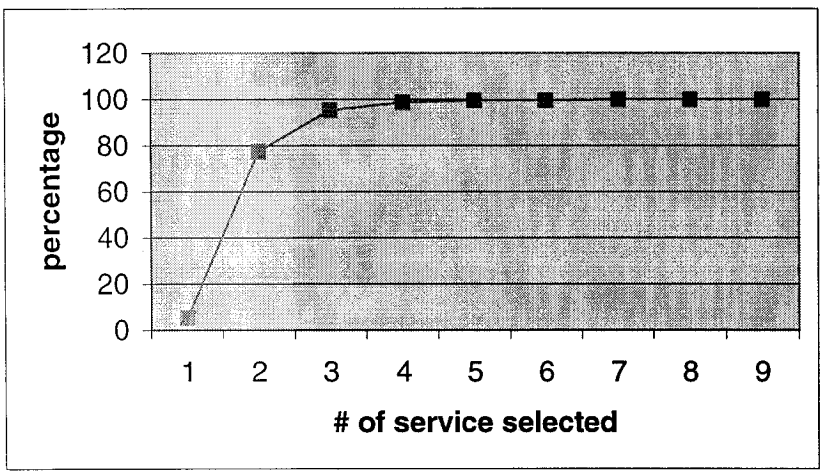

Figure 5.4 Overall simulation results of experiment 2

5.3.3 Experiment 3: *the selection criteria are very high; most of the composite services do not satisfy the user's requirements*

In this experiment, the user selection criteria are much higher than experiment 1, which makes most of the composite services do not satisfy the user's requirement. Comparing the *latency* parameter from Table 5-7 and 5-8, we can see that more than half of the elementary services violate the user selection criteria. In other words, only a small portion of the overall composed services satisfies the user's requirements. Table 5-9 summarizes the overall simulation results of experiment 3.

Table 5-7: QoS parameters for each service community in experiment 3

	Throughput (Kbps)	Latency (ms)	Reliability (%)	Exec. Time (ms)	Price (dollar)
C2E	100 ~ 200	1 ~ 20	80 ~ 100	5 ~ 15	50 ~ 100
E2F	100 ~ 200	1 ~ 20	80 ~ 100	5 ~ 15	50 ~ 100
F2G	100 ~ 200	1 ~ 20	80 ~ 100	5 ~ 15	50 ~ 100
C2F	100 ~ 150	1 ~ 30	80 ~ 100	10 ~ 20	100 ~ 150
E2G	100 ~ 150	1 ~ 30	80 ~ 100	10 ~ 20	100 ~ 150

Table 5-8: user's selection criteria for the composite service in experiment 3

	Throughput (Kbps)	Latency (ms)	Reliability (%)	Exec. Time (ms)	Price (dollar)
Value	≥120	≤8	≥50	≤20	≤200
Weight	1%	20%	5%	4%	70%

Table 5-9: The summary of the experiment results for experiment 3

# of Services in community / # services chosen for composition	1	2	3	4	5	6	7	8	9	10

0	99	99	98	90	83	86	84	78	76	75
1	100	100	100	98	96	93	95	89	88	87
2				98	99	100	99	97	97	95
3				100	100		100	99	100	98
4								100		99
5										100

Table 5-9: The summary of the experiment results for experiment 3 (cont.)

# of Services in community / # services chosen for composition	20	30	40	50	60	70	80	85	Summary
0	26	5	2	0	1	0	0	0	**902 (50.11%)**
1	51	43	39	34	35	37	47	41	**1273 (70.72%)**
2	79	73	70	62	70	69	75	65	**1548 (86.00%)**
3	91	93	81	80	84	83	85	85	**1679 (93.28%)**
4	95	95	90	86	93	93	91	91	**1733 (96.28%)**
5	100	98	95	90	97	97	92	92	**1761(96.28%)**
6		100	98	92	98	100	97	94	**1779 (98.83%)**
7			99	92	98		98	96	**1783 (99.06%)**
8			100	95	99		99	96	**1789(99.39%)**
9				100	99		100	98	**1797 (99.83%)**
11					99			99	**1798(99.89%)**
12					100			99	**1799 (99.94%)**
15								100	**1800 (100%)**

From the above table, we can see that by selecting three services from each service community, we have 93.28% chance to get the optimal solution. 93.28% is not a bad number given the strict selection criteria in this experiment. If you want to increase the selected web services to 5 for each service community, then the chance of getting the best

result will be increased to 97.84% accordingly. This is still acceptable in most cases. The overall results for experiment 3 are illustrated in Figure 5.5.

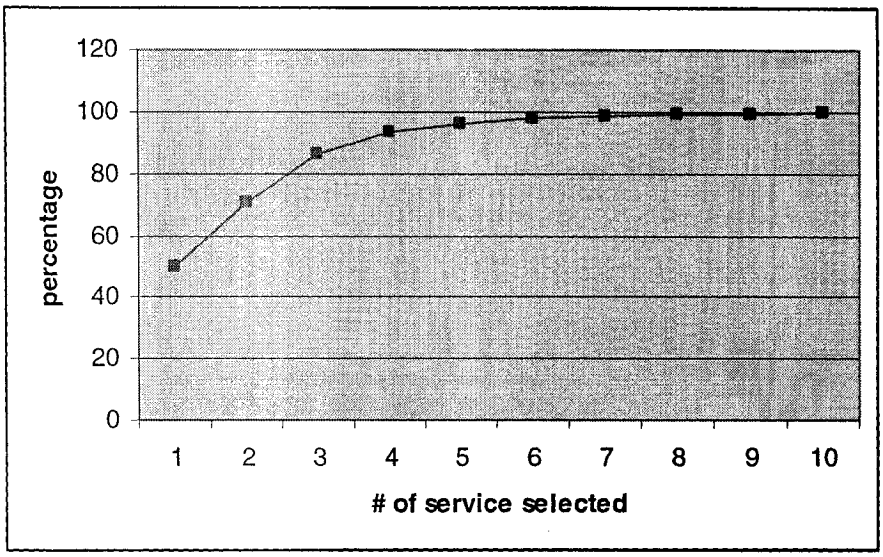

Figure 5.5 Overall simulation results of experiment 3

5.3.4 Experiment 4: *the parameter for 'throughput', 'latency', 'reliability', and 'execution time' are all fixed for each type of services. Only 'price' is randomly changed*

This experiment is designed to test the special case where only one QoS parameter is different for all the elementary services. As shown in Table 5-10, only the parameter *price* can be randomly generated. According to the user requirements in Table 5-11, all web services, either elementary or composite services, satisfy the user requirements for *throughput, latency, reliability*, and *execution time*. The only parameter affects the composed result is *price*.

Table 5-10: QoS parameters for each service community in experiment 4

	Throughput (Kbps)	Latency (ms)	Reliability (%)	Exec. Time (ms)	Price (dollar)
C2E	180	12	85	15	50 ~ 100
E2F	150	14	90	8	50 ~ 100
F2G	160	8	95	10	50 ~ 100

C2F	120	8	88	12	50 ~ 150
E2G	140	10	75	16	50 ~ 150

Table 5-11: user's selection criteria for the composite service in experiment 4

	Throughput (Kbps)	Latency (ms)	Reliability (%)	Exec. Time (ms)	Price (dollar)
Value	≥120	≤35	≥70	≤35	≤200
Weight	5%	10%	5%	10%	70%

Table 5-12: The summary of the experiment results for experiment 4

# Services in community / # services chosen for composition	1	2	3	4	5	6	7	8	9	10
0	45	6	2	1	0	0	0	0	0	0
1	100	100	100	100	100	100	100	100	100	100

Table 5-12: The summary of the experiment results for experiment 4 (cont.)

# of Services in community / # services chosen	20	30	40	50	60	70	80	85	Summary
0	0	0	0	0	0	0	0	0	**54 (3.0%)**
1	100	100	100	100	100	100	100	100	**1800 (100%)**

From the simulation results shown in Table 5-12, we can see that by selecting one web service (which has the minimum cost function value) from each service community, the probability of finding the optimal composite service becomes 100%. As usual, the number of simulation conducted for experiment 4 is 1800.

From this experiment results, we cannot draw the conclusion to say that selecting one

web service from each service community guarantees 100% finding of the optimal result. However, the results of this experiment do demonstrate that our proposed algorithm works better when only one QoS parameter changes.

In this chapter we listed the simulation data and the corresponding diagrams for each experiment. In each section, we gave a short description and analysis. The detailed analysis of our algorithm, including the complexity and fault tolerant issues, are given in next chapter.

Chapter 6

Experiment Results Analysis

Based on the experimental results listed in previous chapter, we will give an overall analysis and draw a conclusion in section 6.1. Besides the effectiveness of the algorithm, fault tolerance is another big challenge. As we discussed in chapter three, sensor networks are highly dynamic and fault tolerant. Therefore, the algorithm used in the area of sensor networks must also be fault tolerant. We discuss this issue is section 6.2. Finally in section 6.3, we analyze the complexity of our algorithm.

6.1 Overall Experiment Analysis

As we said, we conducted four separated experiments with different QoS parameters and different user requirements. The first three experiments are general cases, while the forth experiment is a special case.

The result of experiment four is ideal. It shows that by selecting one web service from each service community and composing them together, we can find the optimal composite service among them. But we have to keep in mind that this conclusion is drawn when only one parameter varies among the elementary web services.

Since the first three experiments all deal with general situations, we combine their results together in Table 6-1 and give a more intuitive view in Figure 6-1. Table 6-1 describes the percentage (see the second row) for getting the optimal composite service when only limited number of web services (see the first row) are selected from each service community and composed together. In total, the number of simulation conducted in these three experiments is 5400, which is large enough to verify the correctness and effectiveness of our proposed approach.

Table 6-1: The combined results from experiment 1, 2, and 3

# of services selected	0	1	2	3	4	5	6	7
Chance of getting best composition (%)	18.44	79.22	93.46	97.35	98.66	99.20	99.59	99.66

Table 6-1: The combined results from experiment 1, 2, and 3 (cont.)

# of services selected	8	9	10	11	12	13	14	15
Chance of getting best composition (%)	99.79	99.94	99.94	99.96	99.98	99.98	99.98	100

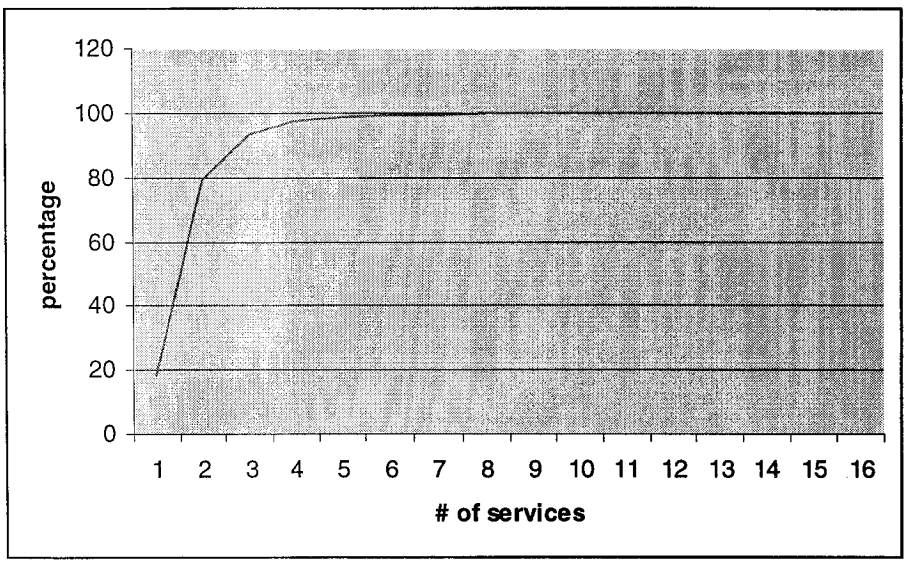

Figure 6.1 Overall results for experiment 1, 2, and 3

Figure 6.1 is a line chart for Table 6-1. The X-axis shows the number of web services selected from each service community, ranging from 0 to 15. The Y-axis gives the percentage for getting the optimal composite solution. From this figure, we can see that the chance of getting the optimal composite service is exponentially increased when the number of elementary services selected is less than 5. When five elementary services are chosen from each service community, the percentage of getting the best composite

service becomes 99.20%. After that, the increasing is linearly, and actually it is almost parallel to the X-axis.

After analyzing the combined results for experiment 1, 2 and 3, we now compare the difference between these three experiments. The summary of these three experiments data is listed in Table 6-2. To make it simple, we only consider the situations where the number of services selected from each service community varies from 0 to 5.

Table 6-2: The comparison between experiment 1, 2, and 3

# of services selected		0	1	2	3	4	5
Chance of getting best composition (%)	Experiment # 1	0	89.56	99.06	99.89	100	
	Experiment # 2	5.22	77.39	95.33	98.89	99.72	99.77
	Experiment # 3	50.11	70.72	86	93.28	96.28	97.84

The above table shows that the results of experiment 1 are a little bit better than those of experiment 2, while the results from experiment 3 are not as good as those of experiment 2. The reason for that is because the user's requirements are different for the three experiments. In experiment 1, all the compositional paths satisfy the user's selection criteria. In other words, the user's selection criteria are very low. In experiment 2, some of the compositional paths do not satisfy the user's requirements, while most of the compositional paths do not satisfy the requester's selection criteria in experiment 3.

As indicated in Table 6-2, with the user's selection criteria become higher and higher, the chance of getting the optimal composite service becomes lower. This is reasonable in some extent, because the higher the requirements the more web services we need to consider. On the other hand, if the requirements are so high that no single elementary service can meet the requirements, then things become very simple because no composition will be necessary. In real life, the clients must have some basic knowledge about the QoS parameters before requesting a composite web service. Therefore, the two extreme situations (very high or very low selection criteria) seldom happen.

In summary, simulations over a wide range of QoS parameters agree with our original idea and serve to verify our algorithm. In other words, our experiment results strongly suggest that the proposed selection algorithm works effectively, i.e., in order to find the optimal composite service, we can just choose several web services from each service community and compose them instead of composing all the possible web services together.

Our experiment result shows that by selecting three web services from each service community and composing them, the percentage of getting the optimal result is 97.35%. If we decide to choose five web services from each service community, then the chance of getting the best solution becomes 99.20%. Even though only one web service is chosen from each service community, there is still nearly 80% opportunity to get the optimal solution.

By using our proposed approach, very limited computation and storage resources will be needed. This makes our algorithm work perfect in situations where only limited resources are available, such as sensor networks. Note there are some tradeoff between accuracy and needed resources. The more web services selected from each service community, the more resources will be needed, and the more accurate our algorithm will be.

6.2 Fault Tolerant Issues

In order to make an algorithm dynamic and robust, the algorithm must be able to adaptive to failure. During the execution of a business process, if one component service fails or becomes overloaded, a mechanism is needed to ensure the running process is not interrupted and the failed service can be quickly and efficiently replaced [6]. In some cases, the failure may happen right before the execution of the business process. In other situations, the execution path may or may not be recalculated according to the user requirements and the running status.

Different strategies are needed for different situations. In this section, three new concepts

are introduced, which are *backup path*, *secondary path*, and *alternative path*. For each concept, we first give its definition. Then the same scenario as stated in Chapter five is used to further depict it. After that we discuss how to generate theses paths using our algorithm.

6.2.1 Backup Path

> **Backup Path:** *For each service component Si on the optimal path, a backup path needs to be generated. The backup path for Si is the optimal path in the original system without Si.*

A backup path is also called replacement path. It is calculated before the execution of a business process. Each service component Si on the optimal path corresponds to a backup path. The backup path for Si is used to replace the optimal path in case of Si fails or becomes unavailable.

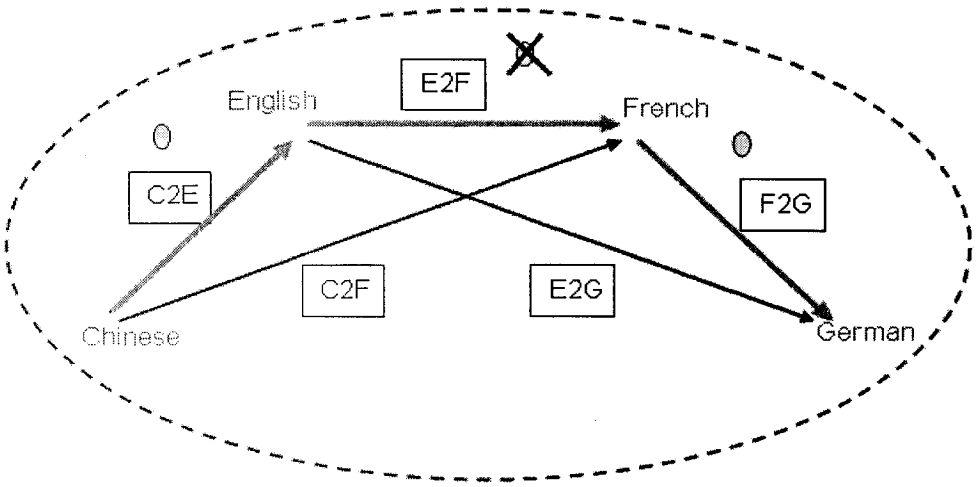

Figure 6.2 Backup path demonstration

Figure 6.2 is an illustration scenario for backup path. In this diagram, the green arrow line shows the optimal path. In this optimal path, there are three service components that are marked by the small filled oval. For each of these service components, a backup path needs to be calculated beforehand. A backup path for Si is generated as usual except that Si is neglected from the calculation. The dotted oval means we need to consider the whole scenario when calculating the backup path.

In our algorithm, generating the backup path for Si is very simple and efficient. We only need to rerun our algorithm without considering Si as an input service. If you want to make the result more accurate, you can add one more web service from the same service community of Si as input when deleting the component Si.

In our scenario, suppose we decide to select three services from each service community and compose them. In order to calculate the backup path for the E2F service on the optimal path, the input from E2F service community becomes two. We can choose to add one more web service from E2F service community as input. Therefore, the total number of services from E2F service community is still three.

6.2.2 Secondary Path

> **Secondary Path:** *For each service component Si on the optimal path, a secondary path needs to be generated. The secondary path for Si is generated from the current service component to the end of the business process without considering its immediate successor.*

A secondary path is also calculated before or during the execution of a business process. A secondary path is needed in case a service component's immediate successor fails or becomes unavailable during the execution. Suppose Si is a service component on the optimal path. The secondary path for Si is an optimal path from the current position to the end of the process. It is generated starting from the current component and its immediate successor is neglected from the calculation.

An example scenario for calculating the secondary path is shown in Figure 6.3. In this diagram, there are three service components on the optimal path. Suppose the E2F service component fails during the execution. Then we need to find another optimal path starting from *English* and ending at *German* without considering the failed E2F service. The dotted oval in the diagram shows the covering area when generating the secondary path.

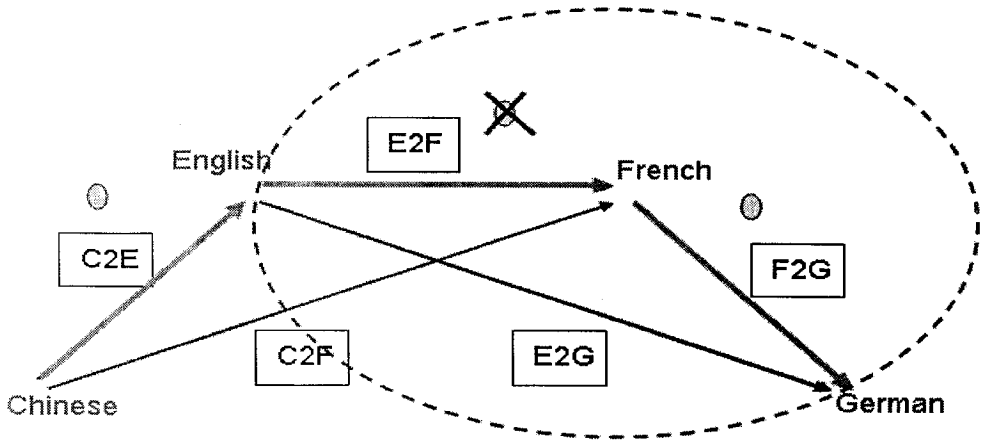

Figure 6.3 Secondary path demonstration

In our algorithm, generating the secondary path for Si is as simple as generating the optimal path. What we need to do is rerun our algorithm. The only difference is the input domain becomes smaller and the immediate successor of Si will not be considered. Again, if you want to make the result more accurate, you can add one more web service from the service community of Si's successor as input when deleting Si's immediate successor.

6.2.3 Alternative Path

> **Alternative Path:** *For each service component Si on the optimal path, an alternative path needs to be generated. The alternative path for Si is generated by selecting an alternative service from the same service community.*

An alternative path is generated during the execution of the business process. The alternative path is needed in case a service component fails and we do not want to change the execution path. Suppose Si is a service component on the optimal path. The alternative path for Si is generated by choosing another service from the same service community of Si.

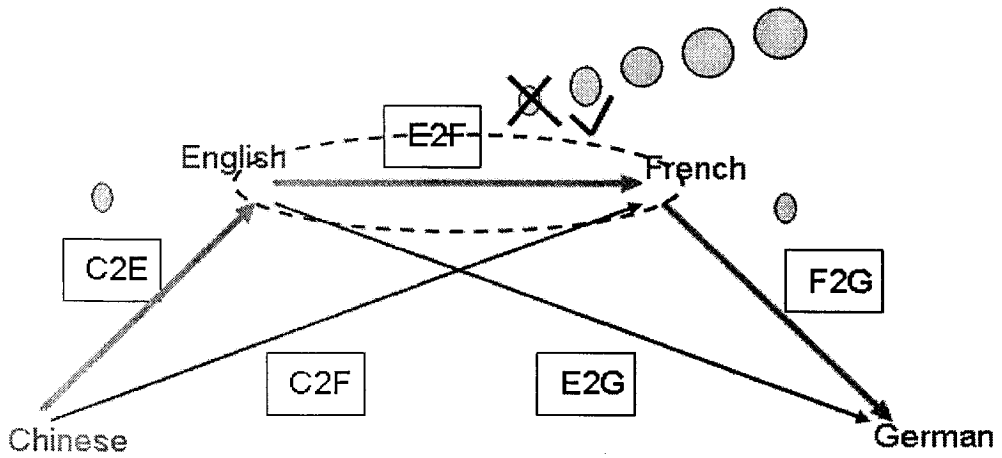

Figure 6.4 Alternative path demonstration

Figure 6.4 shows an illustration scenario for generating the alternative path. In this diagram, suppose the E2F service component on the optimal path fails during the execution and we do not want to change the execution path. Then we only need to choose another E2F service to replace the failed one. Of course, the chosen E2F service has the best performance (i.e. with minimum cost function) among all the E2F services except the failed one. As stated previously, the dotted oval in the diagram shows the covering area when generating the secondary path.

In our algorithm, generating the alternative path for Si is very straightforward. We only need to select another web service to replace Si from the same service community.

In summary, for all these three cases discussed above, we can just simply rerun our algorithm with different input parameters in order to recover from the failure. The calculation of the corresponding paths can be done either before the execution or during the execution, depending on the user requirements. Since our proposed algorithm is very simple and efficient, it's very easy to generate the ***backup path***, ***secondary path***, and *alternative path* for each service component. The detailed complexity analysis for our proposed algorithm is discussed in the next section.

6.3 Complexity Analysis

Complexity is always an important factor to be considered when designing a protocol or

algorithm. In one hand, the resources in a sensor node are very limited. On the other hand, the number of available web services keeps increasing. Therefore, our proposed algorithms must be efficient enough to cater this change.

The complexity analysis of our proposed approach is divided into the following three parts. We first analyze the complexity of the elementary service selection algorithm 4-1. Then the complexity of algorithm 4-3, *selecting the service components to construct an optimal path*, is examined. Finally, we give the complexity analysis for calculating the fault tolerance paths.

6.3.1 Complexity of Elementary Service Selection Algorithm 4-1

Suppose N_i is the number of web services in service community i and let N = max $\{N_1, N_2 \ldots N_k\}$, where k is a constant value and represents the total number of service community we have. Then the complexity of algorithm 4-1, *selecting the best elementary service*, is analyzed as follows.

- Selecting all the services that satisfy the requirements – O(N)
- Calculating the average value of each QoS parameter – O(N)
- Computing the cost function for all web services – O(N)
- Sorting each service community according to cost function – O(NlogN)
- Choosing the optimal elementary service – O(1)

In the first glance, we can see that the complexity of algorithm 4-1 is O(NlogN), because all these steps listed above are executed in sequential order. As we can see, the complexity O(NlogN) comes from the step of sorting web services in each service community. However, further analysis raises the following question. Do we really need to sort all the services in order to get the best one?

In our algorithm 4-1, we only need to select the best web service from each service community in terms of cost function. Instead of sorting all the services, we can simply put one service into a buffer and compare its cost function with the services thereafter. In this way, we can improve the complexity of algorithm 4-1 to O(N).

6.3.2 Complexity of Composite Service Selection Algorithm 4-2

The complexity of algorithm 4-3, *selecting the service components to construct an optimal path*, is analyzed as follows.

- Selecting all the services that satisfy the requirements – O(N)
- Computing the cost function for all satisfied web services – O(N)
- Selecting a few best performance services from each service community – O(N)
- Composing the constant number of web services together and finding the optimal path – O(1)

Note, in the step of selecting a few best performance services from each service community, we only need to put a few (for example five) services into a buffer and compare their cost functions with the other services. The complexity of this mechanism is O(N). Therefore, the overall complexity of our algorithm 4-3 is O(N).

6.3.3 Complexity for Calculating the Fault Tolerant Paths

For generating the alternative path, we only need to pick the second best service from the same service community, so the complexity of that is O(1).

In order to calculate the backup path and secondary path, we need to rerun our selection algorithm 4-3 with different input parameters. Therefore, the complexity of that is O(N). However, before we rerun the algorithm, the cost functions for the web services have been computed and a few of the best services have already been selected. We do not need to redo those steps. Therefore, the complexity of generating these replacement paths is actually O(1).

In brief, our proposed approach for constructing the optimal path is very simple, fast, and efficient. It achieves the best complexity result, O(N). Since N is the number of services in each service community, no other service selection algorithm can achieve a better complexity than ours.

Note that although this work has been motivated by coverage problems in sensor networks domain, the results obtained are generally applicable to all other domains. By using our proposed service selection algorithm, only limited computation and storage resources are required. This will consequently make the web services composition process faster.

Chapter 7

Conclusions and Future Work

We have designed resource-efficient service selection algorithm that can be used in the area of sensor networks. The first part of this thesis addresses the problem of integrating web service technique to sensor networks domain. The focus here is on web service composition. In the second part of this thesis we build a dynamic, simple, and fault tolerant web service selection algorithm that only consume limited computation and storage resources. In this chapter, we summarize our research and provide direction for future work.

7.1 Conclusions

Nowadays, sensor networks are comprised of a large number of sensor nodes that are distributed and co-ordinate together to perform some sensing tasks. They have applications in a variety of fields such as environmental monitoring, military purposes, and gathering sensing information. Incorporating web service technology to sensor networks domain is an emerging and promising research area that will make sensor networks more popular and ubiquitous.

The quality-based selection of web services is an active research topic in the dynamic composition of web services. In our research, we introduce the concept of cost function, which can be easily calculated by using the non-functional parameters such as service price, execution time, reliability, network latency, and throughput. These parameters can either be dynamically calculated or provided by each web service, or withdrawn from third party agent. Obviously, the needed QoS parameters can be extended according to users' requirements without affecting our model for calculating cost function.

After sorting the elementary services by their cost function, we need to find the optimal composite service. Combining the best elementary services together does not guarantee to

get the optimal composite service. Therefore, an efficient web service selection technique is required. The mechanism we used is choosing only a few elementary services from each service community. Then we compose them together to find the optimal composite service.

From our experiment result, we can see that by selecting three elementary services from each service community, we have 97.35% chance for finding the optimal solution in all 5400 experiments. The result will be better if more than three web services are chosen from each service community. However, the more web services are chosen, the more system resources are needed. So we need to consider the tradeoff between the accuracy and the available resources. A separated group of experiment (experiment 4) shows that if we fix all QoS parameters except one, only one web service needed from each service community in order to get the optimal composite service.

Our proposed service selection algorithm has the following characteristics:

- It is resource efficient. The proposed selection algorithm only consumes limited system resources in comparison with other service selection algorithms. Based on the available resources, you can determine the number of services needed from each service community in order to get the optimal composite service.
- It is simple. The proposed algorithm has complexity of O(N), which is the best algorithm that can be achieved.
- It is dynamic. The simplicity of our proposed algorithm makes dynamic web services composition very easily.
- It is fault tolerant. The proposed algorithm is adaptive to failure. This caters the failure-prone characteristics of sensor networks and the rapidly changed characteristics of web services world.

The conclusion we draws from our experiments makes dynamic web service composition possible in sensor networks applications. Moreover, the proposed service selection algorithm can also be used in other situations to save system resources and accelerate the execution time.

7.2 Future Work

The proposed resource-efficient web service selection algorithm provides the possibility of integrating and composing web services in sensor networks domain. However, there is still a long way to go in the commercialization process. Some work remains to be done in future research.

First, in our implementation, the web services QoS parameters are randomly generated based on the given range. This is reasonable because different web services have different parameters. In order for our proposed algorithms work in real life, we have to design and implement a QoS-based agent which can be used to dynamically and automatically calculate the nonfunctional parameters for each web service.

Second, as stated in section 4.2, the whole web services composition consists of four steps: service discovery, service selection, service binding, and service execution. Our research only focuses on the service selection step. Therefore, we need to integrate our selection algorithm into one of the web services composition frameworks. What kind of composition frameworks we choose is really depends on the application domains. Hence, more research should be done in this area.

Last but not least, in order for our proposed algorithms work properly in sensor networks domain, we need to set up a web service based architecture for sensor networks. Reference [45] is a very good point to start with.

References

[1] S. Dustdar and W. Schreiner, "A Survey on Web Services Composition", International Journal of Web and Grid Services, 1, 2005

[2] Y. Liu, A. Ngu, and L. Zeng, "QoS Computation and Policing in Dynamic Web Service Selection", In *Proceedings of the 13th international World Wide Web conference*, pp66-73, 2004

[3] N. Milanovic and M. Malek, "Current Solutions for Web Service Composition", IEEE Internet Computing, v.8, pp51-59, 2004

[4] S. Ran, "A Model for Web Services Discovery with QoS", ACM SIGecom Exchanges, 4(1):1-10, 2003

[5] B. Srivastava and J. Koehler, "Web Service Composition – Current Solutions and Open Problems", ICAPS 2003 Workshop on Planning for Web Services, 2003.

[6] T. Yu, and K. Lin, "A Broker-Based Framework for QoS-Aware Web Service Composition", In Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE'05), pp22-29, 2005

[7] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng, "Quality Driven Web Service Composition", In *Proceedings of the 12th international conference on World Wide Web(WWW)*, May 2003

[8] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. "QoS-Aware Middleware for Web Services Composition", IEEE Transactions on Software Engineering, pp311-327, Vol. 30, No.5 May 2004

[9] T. Yu and K. J. Lin, "Service Selection Algorithm for Web Services with End-to-end QoS Constraints", In *Proceedings of the IEEE International Conference on E-Commerce Technology(CEC'04)*, pp129-136, 2004

[10] Daniel A. Menasce, "QoS Issues in Web Services", IEEE Internet Computing, 6(6), 2002

[11] Wil M.P. van der Aalst, M. Dumas, and A. H.M. der Hofstede, "Web Service

Composition Language: Old Wine in New Bottles?", In *Proceedings of the 29th Conference on EUROMICRO*, pp298, Sept. 2003

[12] K. Fujii and T. Suda, "Dynamic Service Composition Using Semantic Information", ICSOC'04, November 15--19, 2004, New York, USA,

[13] B. Medjahed, A. Bouguettaya, A. Elmagarmid, "Composing Web Services on the Semantic Web", the VLDB journal, Volume 12, Issue 4, pp.33--351, Nov. 2003.

[14] Z. Maamar, S. Kouadri, and H. Yahyaoui, "A Web Services Composition Approach based on Software Agents and Context", In *Proceedings of the 2004 ACM symposium on Applied computing*, pp1619--623, 2004

[15] E. Sirin, J. Hendler, and B. Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions", In *Proceedings of the ICEIS-2003 Workshop on Web Services: Modeling, Architecture, and Infrastructure*, 2003

[16] S. Ponnekanti and A. Fox, "SWORD: A Developer Toolkit for Web Service Composition", In *Proceedings of the 11th International World Wide Web Conference*, 2002.

[17] X. Su and J. Rao, "A Survey of Automated Web Service Composition Methods", In *Proceedings of First International Workshop on Semantic Web Services and Web Process Composition*, July 2004

[18] A. Sheth, J. Cardoso, J. Miller, and K. Kochut, "QoS for Service-oriented Middleware", In *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI02)*, July 2002

[19] M. De Backer, G. Dedene, and J. Vandenbulcke, "Web Services Composition, execution, and visualization", In *Proceedings of the 12th IEEE International Workshop on Program Comprehension*, pp264, 2004

[20] S. Huang, X. Wang, and A. Zhou, "Efficient Web Service Composition Based on Syntactical Matching", In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pp782-783, 2005

[21] M. Carman, L. Serafini, P. Traverso, "Web Service Composition as Planning", In

proceedings of ICAPS03 International Conference on Automated Planning and Scheduling, Trento, Italy, 2003

[22] S. Ae Chun, V. Atluri, and N. R. Adam, "Policy-based Web Service Composition", In *Proceedings of the 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications* (RIDE'04), pp85-92, 2004

[23] M.C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "QoS Aggregation for Web Service Composition using Workflow Patterns", In *Proceedings Eighth IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pp 149-159, Sept. 2004

[24] S. Weerawarana and F. Curbera, "Business Process with BPEL4WS: Understanding BPEL4WS, Concepts in business processes", available at http://www-128.ibm.com/developerworks/webservices/library/ws-bpelcol1/, last visited on July 15, 2005

[25] "OWL-S: Semantic Markup for Web Services", available at http://www.w3.org/Submission/OWL-S/, last visited on July 15, 2005

[26] "XLANG -- Web Services for Business Process Design", available at http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, last visited on July 15, 2005

[27] "Web Services Flow Language (WSFL 1.0)", available at http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, last visited on July 15, 2005

[28] M. Keen, J. Cavell, S. Hill, C. K. Kee, W. Neave, B. Rumph, H. Tran, "BPEL4WS Business Processes with WebSphere Business Integration: understanding, Modeling, Migrating", IBM Red Book, available at http://www.redbooks.ibm.com/redbooks.nsf/0/9c602efe338cda6f85256ebb00471cf4?OpenDocument, last visited on July 15, 2005

[29] "Web Service Choreography Interface (WSCI) 1.0", available at http://www.w3.org/TR/wsci/, last visited on July 15, 2005

[30] "Business Process Management Initiative", available at http://www.bpmi.org/, last visited on July 15, 2005

[31] Ethan Cerami, "Web Services Essentials", Book, CA, February 2002.

[32] J.Roy and A. Ramanujan, "Understanding web services", IEEE IT professional Magazine, vol. 3, no. 6, pp. 69-73, November 2001

[33] R. Nagappan, R. Skoszylas, and R. Sriganesh, "Developing Java Web Services", Wiley Publishing Inc, Indianapolis, Indiana, 2003

[34] "Simple Object Access Protocol (SOAP) 1.1", available at http://www.w3.org/TR/soap12-part1/, last visited on July 15, 2005

[35] "Web Services Description Language (WSDL) 1.1", available at http://www.w3.org/TR/wsdl, last visited on July 15, 2005

[36] "UDDI Version 3 Feature List", available at http://www.uddi.org/pubs/uddi_v3_features.htm, last visited on July 15, 2005

[37] "W3C working draft of the web services architecture specification", available at http://www.w3.org/TR/2003/WD-ws-arch-20030808, last visited on July 15, 2005

[38] "Web Services Architecture Overview – the next stage of evolution for e-business", IBM Web Services Architecture Team, available at http://www-128.ibm.com/developerworks/webservices/library/w-ovr/?dwzone=webservices, last visited on July 15, 2005

[39] H. Kreger, "Web Services Conceptual Architecture (WSCA 1.0)", May 2001, IBM software Group

[40] G. Alonso, et al., "Web Services: Concepts, Architectures, and Applications", NY Springer, 2004

[41] "XML Web Services and SOA Guidebook: Glossary", available at http://www.reactivity.com/guidebook/glossary.html, last visited on July 15, 2005

[42] "Web Services Architecture", available at: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#id2260892k, last visited on July 15, 2005

[43] R. Mogha and V. V. Preetham, "Java Web Services Programming", M&T Books, NY, 2002

[44] F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Sensor Networks," *IEEE Communications Magazine*, vol. 40, no. 8, pp102--114, 2002

[45] F. C. Delicato, P. F. Pires, L. et al , "A flexible web service based architecture for wireless sensor networks", In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pp730, 2003

[46] K. Sohrabi et al., "Protocols for Self-Organization of a Wireless Sensor Network," *IEEE Personal Communications*, vol. 7, no. 5, pp. 16-27, Oct. 2000

[47] A. Woo, and D. Culler, "A transmission Control Scheme for Media Access in Sensor Networks," In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pp. 221--235, 2001

[48] L. Li, and J. Y. Halpernn, "Minimum Energy Mobile Wireless Networks Revisited." In *Proceedings of IEEE International Conference on Communications (ICC)*, June 2001

[49] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for Information Dissemination in Wireless Sensor Networks," In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pp174--185 1999

[50] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks." In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pp56--67, 2000

[51] K. Sohrabi et al. "Protocols for Self-Organization of a Wireless Sensor Network," *IEEE Personal Communications*, vol. 7, no. 5, pp16-27, Oct. 2000

[52] C. Shen, C. Srisathapornphat, and C. Jaikaeo, "Sensor Information Networking Architecture and Applications", *IEEE Personal Communications*, Vol. 8 no. 4 pp52-59, Aug. 2001

[53] H.O. Marcy, J.R. Agre, C. Chien, L.P. Clare, N. Romanov, and A. Twarowski, "Wireless sensor networks for area monitoring and integrated vehicle health management applications", In *Proceedings of the AIAA Guidance, Navigation, and Control Control Conference and Exhibit*, August 1999

[54] B.G. Celler, T. Hesketh, W. Earnshaw, and E.D. Ilsar, "An instrumentation system for the remote monitoring of changes in functional health status of the elderly at home". In *Proceedings of the 16th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, November 1994

[55] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao, "Habitat monitoring: application driver for wireless communications technology", In *Proceedings of the ACM SIGCOMM Workshop on Data Communication in Latin America and the Caribbean*, April 2001

[56] A. Bharathidasan, V. A. Ponduru, "Sensor networks: an overview", Department of Computer Science, University of California, Davis, 2001

[57] "W3C XML Protocol Working Group Charter", available at: http://www.w3.org/2005/02/XML-Protocol-Charter, last visited on July 15, 2005