Towards Integration of SOAP-Based Web Services and OGC Web Services

Shujing Shu
University of New Orleans

TOWARDS INTEGRATION OF SOAP-BASED WEB SERVICES AND OGC WEB SERVICES

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
The Department of Computer Science

by

Shujing Shu

B.S., Peking University, China, 1995

May 2004

ACKNOWLEDGEMENT

I started this thesis with little knowledge about general web services and OGC web services. Here I am, after finishing my thesis, writing this acknowledgment. This thesis is the result of several months of hard work whereby I have been accompanied and supported by many people. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them.

The first person I would like to thank is my direct advisor, Dr. Shengru Tu. The many discussions we had really helped me make the right decisions on many issues I encountered during this research. His great enthusiasm and integral view on research has made a deep impression on me. Besides the many valuable advices and insightful comments and instructions, his endless patience and kindness are always appreciated.

I would like to thank Dr. Tu's graduate student Ying Wu, and visiting student Vianney Bizot. Both of them have answered my questions with so much patience. Without their help, I would not have mastered lots of new knowledge so quickly.

I would like to thank my thesis committee members, Dr. Ming-Hsing Chiu and Dr. Golden G. Richard III, for their priceless instructions, help and guidance on my graduate study. Both of them have been so kind to me. Thank you!

I would like to give special thanks to Dr. Andre Skupin. Being his research assistant, besides the GIS knowledge I learnt from him, I also learnt how to design and conduct a research step by step systematically and logically. His great support in my thesis project is greatly appreciated.

I am very grateful for my husband Xueyan, for his love and patience during the graduate study period at The University of New Orleans. One of the best experiences that we lived through in this period was the birth of our son Nathan, who provided an additional and joyful dimension to our life mission.

Finally, I would like to share this accomplishment with my parents and my grandfather in China. Without their love and support, this is impossible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Over the last several years, the Web Services model of Geographic Information Systems has been rapidly evolved and materialized. In this thesis project, I have reviewed the current status of integrating the general Web Services technology (SOAP, WSDL, and UDDI) and OpenGIS Consortium (OGC) Web services standards in developing distributed GIS computing.

The overlap of the web service model and the technology stack between the SOAP-based Web Services and OGC Web Services indicates the feasibility of integration. OGC has named all core general Web Services Technologies (SOAP, WSDL, UDDI) in its envisioned OWS architecture. OGC has also started putting efforts for the integration by conducting experiments, which include a SOAP experiment and an UDDI experiment. However, these experiments only identified some very specific issues based on small number of testing interfaces and scenarios.

There are leading GIS software vendors who have adopted both areas in their implementation. The ESRI ArcWeb Services is a good example, which implements OGC Web Services Interfaces using SOAP, WSDL, and UDDI.

In my implementation experiment, Java Web Services Developer Pack is used to build a client of Microsoft TerraService. SOAP messages are constructed to retrieve DOQ images from the TerraService as the background to display ArcSDE feature data. Query functionalities on the feature data are implemented.

CHAPTER 1 INTRODUCTION

Over the past several years, the Web Services model of Geographic Information Systems (GIS) has been rapidly evolved and materialized [1, 12, 20]. This distributed GIS model is based on independently provided, specialized, interoperable GIS Web Services. The rapid development of GIS Web Services model owes in part to the advancements in general Web Service technologies (the SOAP-based Web Services), and in part to the focused efforts by the OpenGIS Consortium (OGC; www.opengis.org) to initiate and develop interoperable GIS Web Service interfaces [1]. The beautify of the Web Services Model is that GIS users can use the services and data provided by the Web Services to meet their purpose, without having to install, learn, or pay for any unused functionalities. In the rest of this thesis, I will use 'general Web Services' and 'the SOAP-based Web Services' interchangeably to refer to general IT Web Services.

Web Services have emerged and developed to provide a systematic and extensible framework for application-to-application interaction, which are built on top of existing Web protocols and based on open XML standards. Industry specifications were submitted to the W3C (e.g. SOAP [21, 22, 23] and WSDL [32]). The industry Universal Description, Discovery and Integration (UDDI) project immerged [26], and recently the Web Services Interoperability Organization (WS-I) was founded [33].

Meanwhile, the OpenGIS Consortium (OGC), representing the GIS industry, has been focusing on Internet GIS by developing specifications like the Web Mapping Server (WMS) and the Web Feature Server (WFS) and by setting up GIS interoperability initiatives [18]. OGC is an international industry consortium of more than 250 companies, government agencies and universities participating in a consensus process to develop publicly available interface specifications. OGC Web Services are a subset of OpenGIS Specifications which support interoperable solutions that "geo-enable" the Web, wireless and location-based services, and mainstream IT [15].

OGC Web Services allow distributed geoprocessing systems to communicate with each other using technologies such as XML and HTTP. This capability is possible because OGC open standards have established rules for these services to advertise the functionality they provide and how to send service requests. In this manner, OGC Web Services provide a vendor-neutral interoperable framework for web-based discovery, access, integration, analysis, exploitation and visualization of multiple online geodata sources, sensor-derived information, and geoprocessing capabilities. Many software vendors have implemented OGC Web Services Specifications, which include leading GIS software vendors, such as Authodesk, ESRI, MapInfo, Intergraph [18].

In my thesis, the development of GIS Web Services based on both general Web Services technology and OpenGIS Web Services standards is reviewed. The current status of integration of the two areas is reviewed and analyzed. In the progress, common misunderstandings of the overlapping concepts and definitions in the two areas are clarified. To my knowledge, there is no comprehensive peer-reviewed source of GIS Web Services by closely comparing and analyzing general Web Services and OGC Web Services. These two areas have been developing in parallel in terms of time and separately in terms of technology. I believe that the integration of these two areas will bring the greatest benefits to the development of GIS Web services.

My implementation experiment is to demonstrate the beauty and power of Web Services in GIS applications. Namely, without storing any unused data or installing unused functionalities on the local disks, the user can dynamically obtain the desired up-to-date data and functionalities. I have used Java Web Service Developer Pack (JWSDP 1.2) to construct SOAP messages to communicate with Microsoft TerraService (http://terraserver-usa.com/) Web Service. DOQ (Digital Orthophoto Quadrangles) images are retrieved from Microsoft TerraService Web Service as the background to display the feature data managed by a local ArcSDE server. The user can perform queries on the feature data.

CHAPTER 2 BACKGROUND

2.1 General Web Services

In this chapter, I review the SOAP-based Web Services technologies and the OpenGIS Web Services.

2.1.1 What is a Web Service?

Despite the fact that the term Web Services has rapidly gained a lot of momentum, there is no single, universally adopted definition of what is meant by the term Web Services. Several major Web services infrastructure providers have published their definitions for a Web Service. Here are three most widely used definitions in the Web Services area.

The World Wide Web Consortium (W3C) defines a Web Service as the following [31].

> “A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web

service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

IBM has two widely cited definitions of a Web Service. One of the definition describes a Web Service with a certain degree of technical details [14].

"A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. A Web service is described using a standard, formal XML notion, called its service description. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface hides the implementation details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages Web Services-based applications to be loosely coupled, component-oriented, cross-technology implementations. Web Services fulfill a specific task or a set of tasks. They can be used alone or with other Web Services to carry out a complex aggregation or a business transaction."

IBM's tutorial defines a Web Service in a much more succinct manner [8].

> “Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes...Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service.”

This definition is widely cited by research literature on both general Web Service and OGC Web Services [1, 8].

Several important points about general Web Services are worth to be emphasized:

- A Web Service is a programmable application, accessible as a component via standard Web protocols like HTTP, XML and SOAP;
- A Web Service is published, located, and invoked across the Web;
- A Web Service is XML-based standards which enables simplicity, extensibility, and interoperability, programming language and platform independency;
- A Web Service works through existing proxies and firewalls;
- A Web Service has performance problems due to XML overload [6].

2.1.2 The Web Services Architecture

The Service-Oriented architecture (SOA) has been formalized to represent the roles and operations of Web Services [10, 11]. Figure 2-1 illustrates the roles and interactions between the roles of the Web Services architecture.

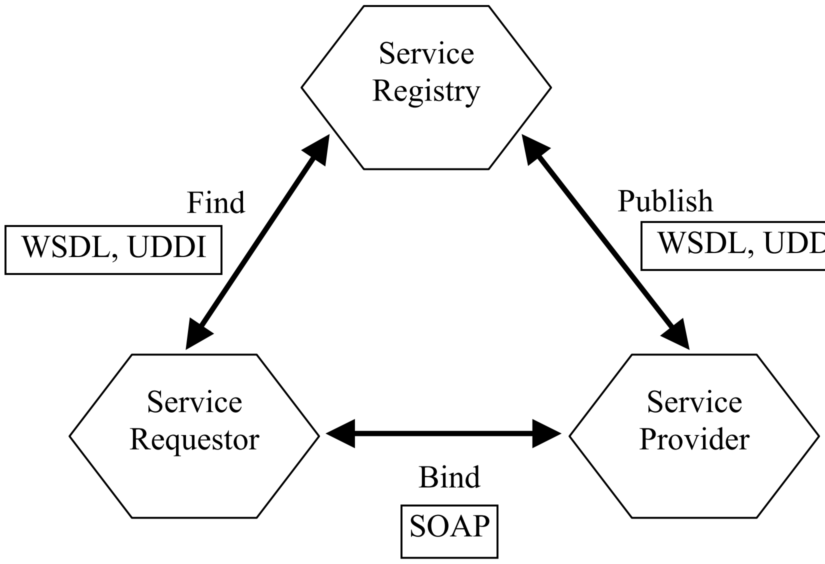


Figure 2-1. The Web Services Architecture.

In this architecture, a service is the implementation of a Web service, which is an interface described by a service description. The service description contains the details of the interface and implementation of the service, which includes its data types, operations, binding information and network location, and so on.

The Web Services architecture contains three roles: service provider, service registry and service requestor. The interactions between the three roles involve three operations: publish, find, and bind.

Service Provider. The service provider is responsible for describing and publishing a service to the service registry. From a business point of view, this is the owner of the service. A service provider typically can be any company that hosts a Web service, which is accessible on some network.

Service Requestor. The service requestor uses a find operation to find a service description locally or published to the service registry, and uses service descriptions to bind with the service provider to interact with the Web service implementation. Any consumer of a Web service, either a browser or a program is a service requestor.

Service Registry. Service registry is the place where service providers publish service descriptions, and it allows service requestors to search for the published service descriptions. It acts as a match-maker between service requestor and service provider. Worth of mentioning, service requestors can obtain service descriptions from sources other than service registry, such as a local file, Web site, FTP site, Discovery of Web Services (DISCO), or Advertisement and Discovery of Services (ADS).

In the Web service architecture, three operations must take place to realize interactions between the three roles: publish, find, and bind.

Publish. Publishing a service description to a service registry makes a service description accessible, so that the service requestor can find it. The complexity of actual details of publish API varies greatly. It can be simply an act of moving the service description into a Web application server's directory structure if the network itself acts as the Service registry. On the other hand, some service registry implementations, such as UDDI, makes a very sophisticated publish operation.

Find. The find operation allows the service requestor to retrieve a service description directly from the service registry, or to query the service registry for the desired type of service. The service requestor can use find operation at design time to retrieve the service's interface description for program development, or at runtime to get the service's binding and location description for invocation.

Bind. The bind operation takes place when a service needs to be invoked. The bind operation allows the service requestor to use the binding details in the service description to locate, contact and invoke the service.

2.1.3 The Web Services Technology Stack

Although theoretically the Web services architecture is independent of any particular standards, interoperability is still a key requirement for large-scale adoption of the architecture. Several key industry leaders (e.g. Microsoft, IBM, and others) have been making efforts to develop a set of XML-based open standards (SOAP, WSDL, UDDI, WSFL) that enable the Web services architecture to be implemented. Here I introduce a conceptual Web Services stack proposed by IBM and Microsoft (Figure 2-2) [11, 14].

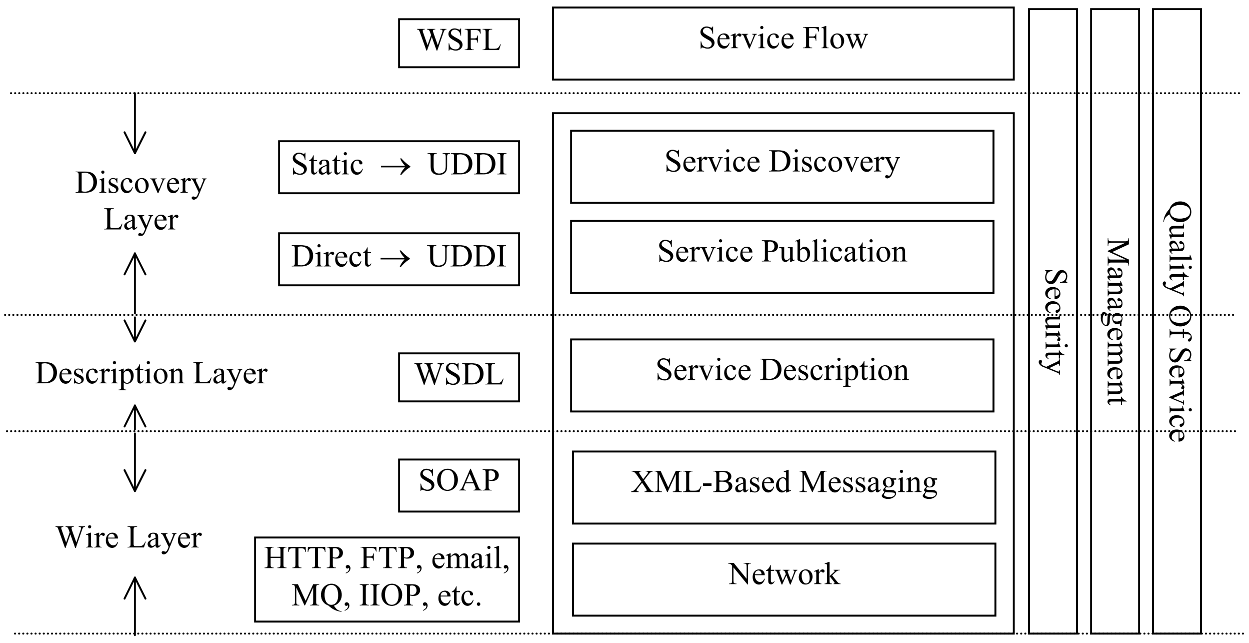


Figure 2-2. The Web Services Technology Stack [14].

This model groups various Web services technologies into three layers: the wire layer, the description layer, and the discovery layer. The upper layers build upon the capabilities provided by the lower layers. The vertical towers represent requirements that must be addressed at every level of the stack. The text on the left represents standard technologies that apply at that layer of the stack. XML is the key technology. Apart from the network protocol everything rests on it.

Wire Layer. The wire layer defines the messaging format and components between the service requestor and the service provider. The base of the wire layer is the network, which makes Web services accessible to a service requestor. Internet-available Web Services use commonly deployed network protocols. HTTP is the de facto standard network protocol for Internet-available Web Services. Other Internet protocols, such as SMTP and FTP, are also supported. Reliable messaging and call infrastructures, such as MQSeries, IIOP, and so on, are supported for Intranet Web Services. SOAP is chosen as the de facto XML messaging protocol for general IT Web services.

Description Layer. The Description layer consists of Web Services description documents. XML is not only the basis of the Wire layer, it is also the basis of service description. Web Services Description Language (WSDL) is the de facto standard for Web Services description in the IT Industry. WSDL defines the interface of a Web service and mechanisms of service interaction, which specifically include the operations supported by the Web service, the input and output of the service, the bindings to concrete network, and data encoding schemes. It is the minimum standard service description necessary to support interoperable Web Services. Additional description is needed to specify other properties of Web services, such as quality of service, service-to-service relationships, and so on. Web Services Flow Language (WSFL) is used to describe Service composition and flow.

Discovery Layer. A Web service needs to be discovered in the first place before it can be invoked. Service discovery closely depends on service publication since a service cannot be discovered if it has not been published.

The simplest and most static service publication is direct publication [14], in which the service provider sending a service description directly to the service requestor. On the other hand, in a complex service publication, a service provider publishes the service description to a local service registry or UDDI service registry.

In parallel to service publication, service discovery can also be either simple and static or complex and dynamic. In a static discovery, the service requestor retrieves a service description from a local file. On the other hand, a service can be discovered at design time or runtime dynamically from a local registry or a UDDI registry.

2.1.4 SOAP, WSDL, and UDDI

The core industry technologies for the general IT industry Web Services for each layer currently are: Simple Object Access Protocol (SOAP) for messaging, WSDL (Web Services Description Language) for description, and UDDI (Universal Description, Discovery and Integration) for publication and discovery.

2.1.4.1 SOAP

Since its introduction in late 1999, SOAP has become the de facto standard for Web services messaging and remote procedure calls (RPCs) [11]. SOAP is a simple, flexible, and highly extensible XML-based messaging protocol [21, 22]. It is programming language, platform, hardware neutral. Rather than defining a new transport protocol, SOAP works on exitsing network protocols, such as HTTP, SMTP, FTP, and so on. Another advantage of SOAP is the ability to pass firewalls, which is essential for a wide area network.

A SOAP message consists of three parts (Figure 2-3).

- An envelope, which provides the framework for packaging message information. It describes what is contained in a SOAP message and how to process it.
- Encoding rules for serializing data.
- An RPC representation that defines how to represent remote procedure calls and responses.

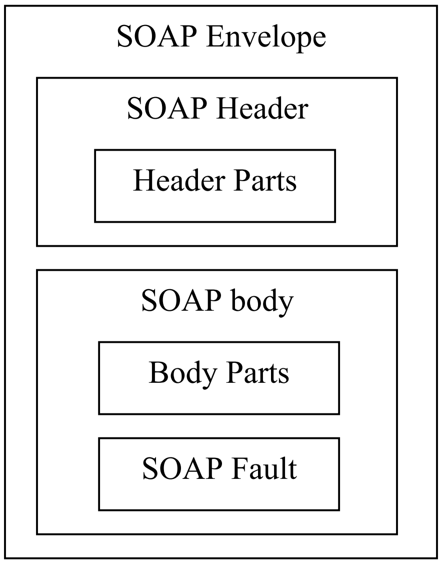


Figure 2-3. SOAP Message Structure.

<u>2.1.4.1.1 SOAP Envelope Framework</u>

The envelope framework is the basic core structure of a SOAP message (Figure 2-3). The root element of the SOAP message is the Envelope element. It defines the various XML namespaces that are used by the rest of the SOAP message. These represent the SOAP specification namespaces for SOAP and schemas for instances, data types, and encoding [21, 22].

The SOAP Envelope element can contain an optional Header element and a mandatory Body element.

Headers are the primary extensibility mechanism in SOAP. The usual information Header element defines includes authentication and authorization, transaction management, routings, and payment processing. The actors attribute specified in the Header element indicate who should which parts of the message, which enables SOAP intermediaries mechanism.

The SOAP body element contains the core information intended for the final message receiver. The Body of the SOAP response can contain a single Fault element as the only child element of the SOAP body to carry SOAP error information. When SOAP is used to perform an RPC, the body contains a single element that contains the method name to be invoked and it's arguments. The namespace of the method name is specified by the web service followed by the type of the target web service. The type of each argument can be optionally supplied using the xsi:type attribute.

2.1.4.1.2 Messaging Using SOAP

There are two types of messaging pattern using SOAP: the Conversational Message Exchanges and the Remote Procedure Calls.

The Conversational Message Exchange is a request-response pattern, in which XML-based content conforming to some application-defined schema are exchanged via SOAP messages. In the simplest case, the user can only send the SOAP request to the service for processing. For example, an airline traveler can send a SOAP request to do an electronic checkin, provided with name of the checkin method and the fromat for encoding the ticket.

Remote Procedure Calls (RPCs) with SOAP is used when there is a need to model a certain programmatic behavior, with the exchanged messages conforming to a pre-defined description of the remote call and its return. To use SOAP for RPCs, you must define an RPC protocol, including [7]:

- how typed values can be transported back and forth between the SOAP representation (XML) and the application's representation (such as a Java class for a ticket), and
- where the various RPC parts are carried (object identity, operation name, and parameters).

The W3C's XML schema specification (www. w3.org/XML/Schema) provides a standard language for defining the document structure and the XML structures' data types. For the typed values, SOAP assumes a type system based on the one in XML schema and defines its canonical encoding in XML. Using this encoding style, the user can produce an XML encoding for any type of structured data. RPC arguments and responses are also represented using this encoding.

SOAP implementations exist for several programming languages, including C, Java, and Perl, which automatically generate and process the SOAP messages. Assuming the messages conform to SOAP specifications, they can thus be exchanged by services implemented in different languages.

2.1.4.2 WSDL

The Web Services Definition Language (WSDL) is an XML-based standard to describe the technical invocation syntax of a Web service. A complete WSDL service description provides two pieces of information: an application-level service description, or abstract interface, and the specific protocol-dependent details that users must follow to access the service at concrete service end points. This separation accounts for the fact that similar application-level service functionality is often deployed at different end points with slightly different access protocol details. Separating the description of these two aspects helps WSDL represent common functionality between seemingly different end points.

An abstract interface can support any number of operations. An operation is defined by the set of messages that define its interaction pattern. For the abstract concepts of message and operation, concrete counterparts are specified in the binding element.

The WSDL schema defines several high level or major elements in the language, which are introduced in the following [11].

PortType – A Web service's abstract interface definition where each child operation elemHoent defines an abstract method signature.

Message – Defines a set of parameters referred to by the method signatures or operations. A message can be further decomposed into parts.

Types – Defines the collection of all the data types used in the Web service as referenced by various message part elements.

Binding – Contains details of how the elements in an abstract interface (portType) are converted into a concrete representation in a particular combination of data formats and protocols.

Port – Expresses how a binding is deployed at a particular network endpoint.

Service – A collection of ports.

So the portType (with details from the message and type elements) describes the what of the Web service. The binding element describes the how, and the port and service elements describe the where of the Web service.

Many developers split their WSDL designs into two parts, each placed in a separate document. The service interface definition, containing the types, message, portType, and binding elements, appears in one file. You can then place this file, for example, on a well-known Web site (on an e-marketplace, for example) for everyone to view. Each organization that wants to implement a Web service conformant to that well-known service interface definition would describe a service implementation definition, containing the port and service elements, describing how that common, reusable, service interface definition was, in fact, implemented at the network endpoint hosted by that organization.

Figure 2-4 outlines how the major elements in WSDL are divided between the service interface definition and the service implementation definition.

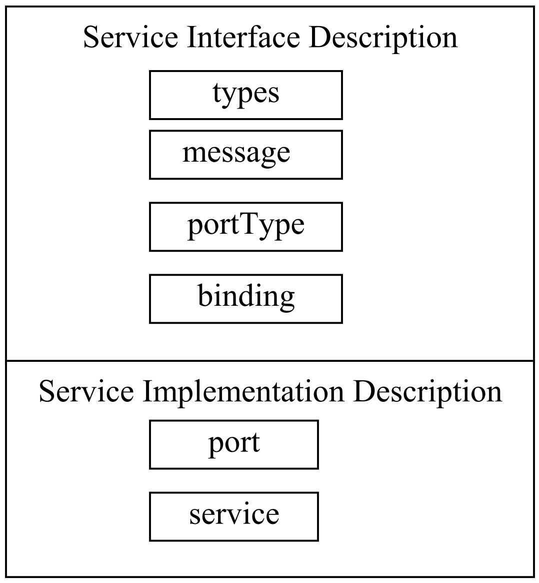

Service Interface Description
types
message
portType
binding
Service Implementation Description
port
service

Figure 2-4. The Key Elements of a WSDL file.

2.1.4.3 UDDI

UDDI is an XML-based registry for Web Services. It defines a way to publish and discover information about services on the Web. The main purpose is to support business to business (B2B) activities. UDDI can be compared to telephone books: it consists of white pages (locate service by name), yellow pages (locate service by type of offering or topic, and green pages (locate service by its characteristics). Every project or company can put its services to the registry by giving information about the product, pricing, etc. No special form for the description of services is required. Though based on

XML, the registry can also describe services implemented in HTML, CORBA, or any other type of programming model or language.

UDDI provides two basic specifications that define a service registry's structure and operation [27, 7]:

- A definition of the information to provide about each service, and how to encode it; and
- A query and update API for the registry that describes how this information can be accessed and updated.

Registry access is accomplished using a standard SOAP API for both querying and updating. Unique keys identify each data entry - businesses, services, and so on - that might be cross-referenced. Theses assigned keys are long hexadecimal strings generated when the entity (in this case, a business) is registered. The keys are guaranteed to be universally unique identifiers (UUIDs).

2.2 OGC Web Services

2.2.1 Overview of OGC Web Services

Within the broader context of Web Services, OGC Web Services (OWS) represent an evolutionary, standards-based framework that enable seamless integration of a variety of online geoprocessing and location services [8]. Even though the most popular definition of OWS is “self-contained, self-describing, modular applications that can be published, located, and invoked across the Web”, OGC Web Services only provide implementation specifications for GIS Web Services applications. Attention must be paid to terminology: when OGC speaks of Web Services, they mean their specifications, which are not equivalent to the general SOAP-based Web Services. This is reasonable, because OGC does not intend to specify general issues that are not specific to Geographic Information.

OWS is one of OGC’s many initiatives for addressing the lack of interoperability among systems that process georeferenced data. In the past several years, OGC has successfully executed several phases of the OWS initiative, including Web Mapping Testbed I (WMT-I) in 1999, WMT II in 2000, OGC Web Service Initiative 1.1 (OWS 1.1) in 2001, and OWS 1.2 in 2002, and produced a set of web-based data interoperability specifications. These specifications include, but are not limited to: Web Map Server (WMS), Web Feature Server (WFS), and Web Coverage Server (WCS).

Each of these specifications defines a number of services. Together these services are referred to as OGC Web Services (OWS). OGC Web Services allow geospatial information to be accessed and processed over the Internet. Geospatial information is generally represented using XML that complies with the appropriate XML Schemas.

2.2.2 OGC Web Services Specifications

Here I will describe three major OGC Web Services implementation specifications, WMS, WFS, and WCS. These documents specify details for the OGC Web Services interfaces. Any software which implements one or more of these specifications should then be able to communicate with any other software that implements the same specification(s).

2.2.2.1 Web Map Service

The Web Map Service (WMS) specification [30] standardizes the way in which clients request maps. Clients requests maps from a WMS instance in terms of named layers and provide parameters such as the size of the returned map as well as the spatial reference system to be used in drawing the map.

The specification defines three WMS operations: GetCapabilities returns service-level metadata, which is a description of the service's information content and acceptable request parameters; GetMap returns a map image whose geospatial and dimensional papameters are well-defined; GetFeatureInfo (optional) returns information about particular features shown on a map.

When requesting a map, a client may specify the information to be shown on the map (one or more "Layers"), possibly the "Styles" of those Layers, what portion of the

Earth is to be mapped (a “Bounding Box”), the projected or geographic coordinate reference system to be used (the “Spatial Reference System”, or SRS), the desired output format, the output size (Width and Height), and background transparency and color.

The OGC Web Map Service (WMS) implementation specification version 1.1.0 defines keyword/value encodings for operation requests using HTTP GET and POST.

2.2.2.2 Web Feature Service

The Web Feature Service (WFS) [29] supports operations such as INSERT, UPDATE, DELETE, QUERY, and DISCOVERY of geographic features. WFS delivers GML representations of simple geospatial features in response to queries from HTTP clients. Clients access geographic feature data through WFS by submitting a request for just those features that are needed for an application. A WFS can either be a basic WFS (a READ-ONLY WFS), which implements the GetCapabilities, DescribeFeatureType and GetFeature interfaces, or a transaction WFS, which, in addition to supporting all the interfaces of a basic WFS, implements the Transaction interface (and optionally the LockFeature interface).

To support transaction and query processing, the following WFS interfaces are defined:

- GetCapabilities: A web feature server must be able to describe its capabilities. Specifically, it must indicate which feature types it can service and what operations are supported on each feature type.
- DescribeFeatureType: A web feature server must be able, upon request, to describe the structure of any feature type it can service.
- GetFeature: A web feature server must be able to service a request to retrieve feature instances. In addition, the client should be able to specify which feature properties to fetch and should be able to constraint the query spatially and non-spatially.
- Transaction: A feature server may be able to service transaction request. A transaction request is composed of operations that modify features; that is create, update, and delete operations on geopraphic features.
- LockFeature: A web feature server may be able to process a lock request on one or more instances of a feature type for the duration of a transaction. This ensures that serializable transactions are supported.

WFS requests, encoded in XML or keyword-value pairs, may be transmitted to a web feature server using either the POST or GET methods. However, the XML encoding is most suitable for packaging as an HTTP POST request and the keyword-value pair encoding is more suitable for packaging as an HTTP GET request.

2.2.2.3 Web Coverage Service

The Web Coverage Service (WCS) [28] supports the networked interchange of geospatial data as “coverages” containing values or properties of geographic locations. Unlike the Web Map Service, which returns static maps (server-rendered as pictures), the Web Coverage Service provides access to intact (unrendered) geospatial information, as needed for client-side rendering, multi-valued coverages, and input into scientific models and other clients beyond simple viewers.

The Web Coverage Service consists of three operations:

- *GetCapabilities*: required operation which returns a description of the service with elements to describe multidimensional data collections from which a coverage may be requested.

- *GetCoverage*: required operation returns values or properties of geographic locations, bundled in a well-known coverage format. Its syntax and semantics are similar to the WMS GetMap request, but several extensions support the retrieval of coverages rather than static maps.

- *DescribeCoverageType*: optional operation returns a description of the structure of the coverages which the WCS returns.

2.3 Microsoft TerraService

The Microsoft TerraServer (http://terraserver.net) web site has been optional since 1998. It stores aerial, satellite, and topographic images of the earth in an SQL database available via the Internet. It is the world's largest online atlas, combining fifteen terabytes of aerial imagery data and 1.5 terabytes of topographic maps from the United States Geological Survey (USGS). The TerraService web service provides a programmatic interface to the TerraServer database. The TerraService web methods are designed to support an OpenGIS compatible Web Map Server application. Microsoft, the USGS, and USDA are members of the OpenGIS.

Briefly, TerraServer is a database repository of high resolution, ortho-rectified imagery that is logically represented as a "seamless mosaic of earth" at several scales (meters per pixel). The TerraService also provides a landmark service that returns place names and a list of all the places within a specified area.

The mosaic is stored as sets of uniformly sized, JPEG or GIF compressed images called tiles. Each tile has a predictable resolution and location on the earth. The TerraService is a "tile service". TerraService methods enable consuming applications to query the TerraServer database of tiles by a number of different methods to determine the existence of tile data over some expanse of geographic territory. Data returned by tile query methods enable a calling application to determine the set of tiles required to build a complete image that covers the queried geographic area. The consuming application can iteratively call TerraService's GetTile() method to actually retrieve the individual tiles, construct a new, larger image, and then crop, re-size, and/or layer other graphic data on-top of the image.

As the way the TerraService is used in this implementation case study, applications typically access it to build an image from multiple TerraServer tiles to use as a background image for some geo-spatial display.

2.4 ArcSDE

ESRI enterprise GIS solution is based on a centralized geo-spatial database. To facilitate spatial data sharing and management, the solution include SDE (Spatial Data Engine) server that acts as an application server. ESRI ArcSDE is an application server that facilitates storing and managing spatial data (raster, vector, and survey) in a DBMS and makes the data available to many kinds of applications. ArcSDE server also maintains the data integrity, manage the transaction, and tune the overall performance of the spatial data service. ArcSDE allows user to manage spatial data in one of four commercial databases (IBM® DB2InformixMicrosoft® SQL Serverand Oracle®). Here, the feature data are managed by ArcSDE on top of an Oracle database.

The ArcSDE server also provides a set of API for customized application development. ArcSDE contains two main application programmer interfaces for developers to build applications that work with and query information contained in multiuser geo-databases: ArcSDE Client API for C programmers and ArcSDE Client API for Java programmers. These APIs provide GIS functions for advanced application development. ArcSDE works as an application server, delivering spatial data to many kinds of applications or even serving spatial data across the Internet.

Since our main purpose here is to demonstrate the functionality of the TerraService web service, the feature data can be from any other sources (e.g., from local disk or from a Web Feature Server). In fact, the implementation is designed for displaying any set of point feature data.

CHAPTER 3 GENERAL AND OGC WEB SERVICES INTEGRATION

As the global players of IT like IBM and Microsoft push the development of the core standards of Web services (SOAP, WSDL, and UDDI), well-known companies have implemented their own Web Services technologies using these standards (e.g., IBM WebSphere, Microsoft .NET, and SUN ONE). The huge impact of these core standards on the development of the SOAP-based Web Services should not be ignored as OGC engages in their Web Services Initiatives. In fact, OGC has started putting efforts on integrating the SOAP-based Web Services technologies into the OGC Web Services framework.

OGC Web Service (OWS) specifications provide standards for implementing interoperable, distributed geospatial information processing software systems (e.g. GIS), by which a user can easily share geospatial data, information, and services with others. However, OGC technology consists mainly of interface specifications and content standards. They do not provide a mechanism for efficiently sharing the distributed computational resources. Meanwhile, the SOAP-based Web Services technology, because of its powerful distributed resource sharing capabilities, has earned tremendous popularity and market. Therefore, the integration of general Web services technology with OWS will greatly benefit the development of GIS Web Services.

In this chapter, I will review the current status of the integration of the SOAP-based web services technologies into the OGC web services infrastructure. First, I will compare the OGC web services architecture with the SOAP-based Web Services architecture. Then I will examine the OGC Web Services Technology Stack to identify the general Web Service technologies already named by OGC in their envisioned infrastructure. After that, I will review the OGC activities of promoting the integration. An example of the integration is also provided. Finally, the system design of my implementation case study is introduced.

3.1 OGC Web Services Architecture

The OpenGIS Web Services architecture defines a common set of interfaces that can be utilized by any application to provide enterprise-wide interoperabilities [19]. Figure 3-1 [13] depicts the Computational Viewpoint of the OGC Web Services architecture for OWS 1.2. At the core of the architecture are network-based, distributed, modular components that perform specific tasks and conform to specific sets of technical specifications that make them interoperable with compatible components. Here only important elements of each component are introduced. The summary description of each element can be found in [19].

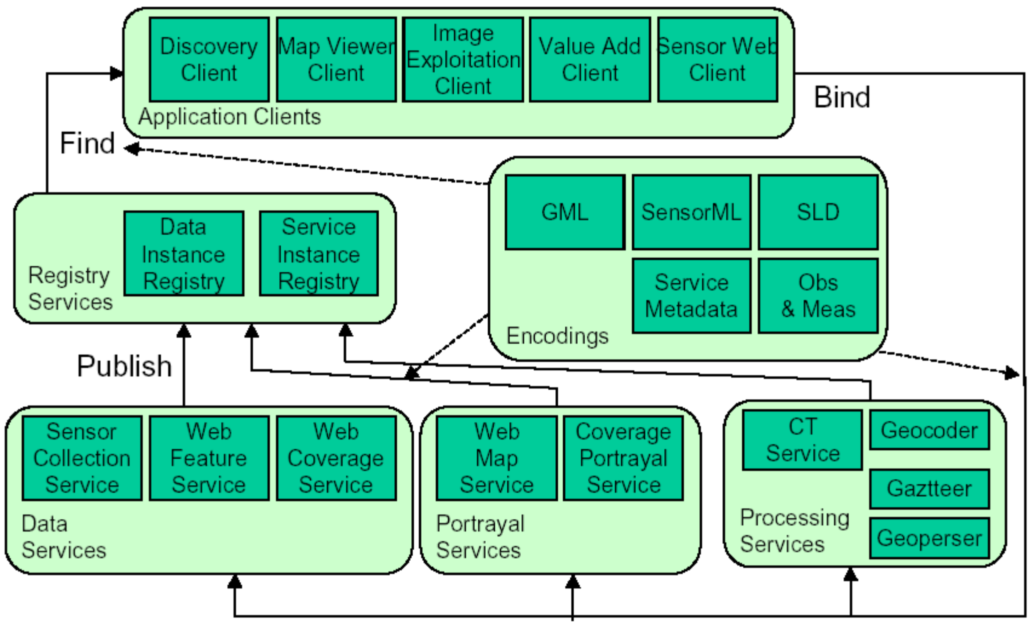


Figure 3-1. OGC Web Services Architecture [13].

Encodings. The encodings describe specialized vocabularies for the transfer of specific kinds of data packaged as messages between application clients and services and between services. Each encoding vocabulary is designed for a specialized use within the architecture. All OWS encoding specifications are application schemas for XML. The most important encoding standard in OGC Web Services is Geography Markup Language (GML) [9]. GML is an XML grammar written in XML Schema for the modeling, transport, and storage of geographic information. GML provides a variety of kinds of objects for describing geography including features, coordinate reference systems, geometry, topology, time, units of measure, and generalized values.

Registry Service. Registry Service provides a common mechanism to classify, register, describe, search, maintain, and access information about network resources (data and services). For OWS1.2, Registry Services include Web Registry Service (WRS), for which the OGC discussion paper is available. Web Registry Server (WRS) interfaces support "one stop shopping" for the registration, metadata harvesting and descriptor ingest, push and pill update of descriptors, and discovery of **OGC** web services types and instances using HTTP as the distributed computing platform.

Data Services. Data Services are the foundational service building blocks that serve data, specifically geospatial data. For OWS1.2, Data Services include Web Object Service (WOS), Web Feature Service (WFS), Sensor Collection Service (SCS), Image Archive Service (IAS) and Web Coverage Service (WCS).

Portrayal Services. Portrayal Services provide specialized capabilities supporting visualization of geospatial information. Portrayal Services are components that, given one or more inputs, produce rendered outputs such as cartographically portrayed maps, perspective views of terrain, annotated images, views of dynamically changing features in space and time, etc.). For OWS1.2, Portrayal Services include Web Map Service (WMS), Coverage Portrayal Service (CPS) and Style Management Service (SMS).

Processing-Workflow Services. Processing-Workflow Services are the foundational application-building-block services that operate on geospatial data and metadata, providing value-add service. For example, the most common type of geo-coding (Geocoder) is converting a street address to a geographic location.

Client Services. Client Services are the client-side components of client applications that interact with users, and on the server-side interact with Server-side Client Applications, Application Servers and Data Servers.

In this architecture, OGC services are categorized according to similar functionalities. Figure 3-1 indicates two important characteristics of OGC services. First, the great complexity of Geographic data determines a relatively large number of different service interfaces are defined. The quality of GIS Web Services depends on an easy interaction between and good interoperability of these services determines. Second, Figure 3-1 implicates that OGC Web services has the same Publish-Find-Bind (PFB) SOA architecture as general web services. It is this SOA paradigm that essentially underpins the Computational and Information Viewpoints of OGC Web Services. In OGC documents, it is called the Service Trading model and the equivalent PFB terminology is broker (service registry), requester, and provider [19]. These two characteristics of OGC Web Services from a certain degree have determined the dependence of OGC Web Services on the powerful general Web Services technology. Thus, integration of these two areas is inevitable.

3.2 OGC Web Services Technology Stack

OGC's Services Interoperability Stack (Figure 3-2) has already named all the core technologies in General IT Web Services [19]. The OGC Services Interoperability Stack is a layered architecture of technology and standards on which services can be implemented and deployed. The lowest levels of the stack enable connectivity of software components by enabling them to bind, send and receive messages. Higher levels in the stack enable interoperability and, via publish-find-bind mechanisms, allow software components to transparently work together in more integrated and dynamic ways.

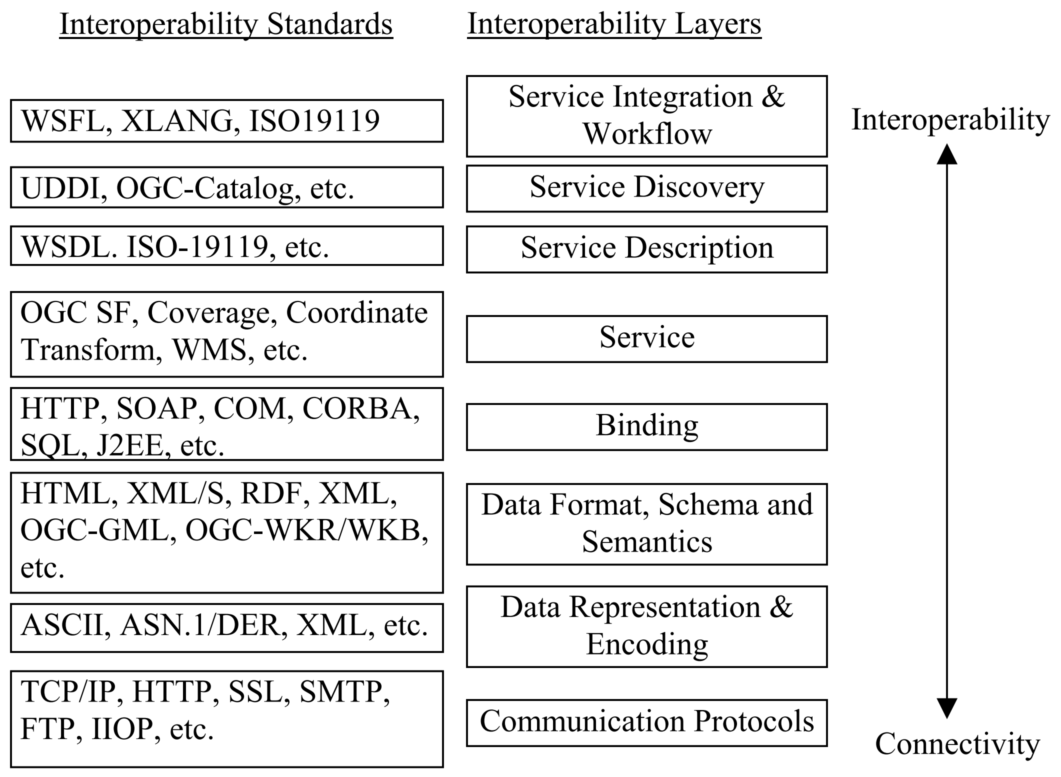

Interoperability Standards
Interoperability Layers
WSFL, XLANG, ISO19119
Service Integration & Workflow
Interoperability
UDDI, OGC-Catalog, etc.
Service Discovery
WSDL. ISO-19119, etc.
Service Description
OGC SF, Coverage, Coordinate Transform, WMS, etc.
Service
HTTP, SOAP, COM, CORBA, SQL, J2EE, etc.
Binding
HTML, XML/S, RDF, XML, OGC-GML, OGC-WKR/WKB, etc.
Data Format, Schema and Semantics
ASCII, ASN.1/DER, XML, etc.
Data Representation & Encoding
TCP/IP, HTTP, SSL, SMTP, FTP, IIOP, etc.
Communication Protocols
Connectivity

Figure 3-2. OGC Web Services Interoperability Stack [19].

This stack overlaps with the general Web Services Stack very well. The overlap of the SOA model and the overlap of the technology stack are strong evidence of the possibility and practicability of porting OGC Web services into General IT Web Services.

3.3 Comparison of the Models of General and OGC Web Services

Figure 3-3 maps general Web Services and OGC Web Services according to the SOA model. As depicted in Figure 3-3, architectural similarity exists between the two areas. Both confirms to the SOA model. Also in terms of technology, as I have mentioned before, OGC names all major technologies (UDDI, WSDL, SOAP) in its envisioned architecture. However, OGC defines and adopts different standards other than those from general Web Services.

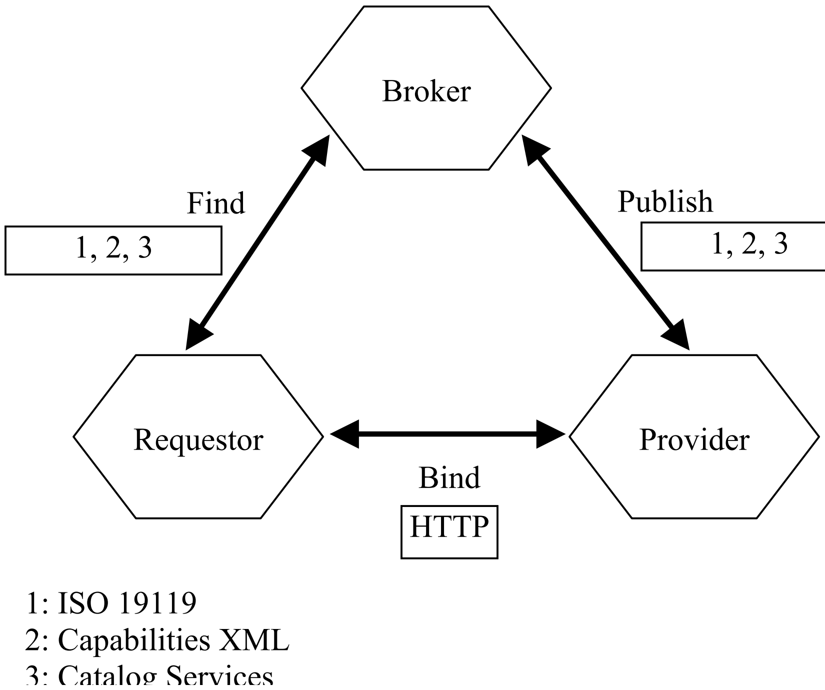


Figure 3-3. Architectural Similarity between General and OGC Web Services [19].

For service registry, OGC has its own registry services, Catalog Services. Catalog services define the set of service interfaces which support organization, discovery, and access of geospatial information. A Catalog can be thought of as a specialized database of information about geospatial resources available to a group or community of users. These resources are assumed to have OpenGIS feature, feature collection, catalog and metadata interfaces, or they may be geoprocessing services. Catalogs have three essential purposes:

- to assist in the organization and management of diverse geospatial data and services for discovery and access,
- to discover resource information from diverse sources and gather it into a single, searchable location, and
- to provide a means of locating, retrieving and storing the resources indexed by the catalog.

OGC employs ISO 19119 and Capabilites XML for its service description. ISO 19119 provides a framework for creating and sharing Geographic Information Services. The ISO service and operation metadata are richer than those of WSDL. Capabilites XML is a format defined by OGC itself. Every OGC Web Service provides an operation “getCapabilities”, which returns a “Capabilities XML” document describing the operations of the service. As with ISO 19119, the service metadata are richer than in WSDL. For example, in ISO 19119, geographic information of the data on which the service operations operate on is provided, such as Geographic BoundingBox and

Geographic Description. However, the operation metadata basically restrict to the signature.

Generally UDDI, WSDL, and Capabilities XML focus on operation signatures; descriptions are available only on service level and appear as free text. ISO 19119 provides free text descriptions not only on service, but also on operation level, and in addition to data types for operation parameters it contains a pointer to permitted values. Although this is a richer description of the context in which the service is used, there is still semantic information missing. The semantic description of service content will provide more powerful service discoveries and more interoperable service interactions.

As to the communication protocol, OGC specified HTTP as the standard communication protocol for OWS, like WMS and WFS. SOAP is on its way of being tested by OGC before it is specified as the standard communication protocol for OWS (see section 3.4).

The inherent complexity of Geographic information may require the necessity of native OGC standards to complement the insufficiency of general Web Services technology to solve GIS problems. The extension of general Web Services technology may also be needed to better satisfy complicated GIS Web Services.

3.4 OGC Integration Activities

Efforts have been made by OGC to integrate general Web services technologies into OGC Web services framework.

The objectives of OWS-1.2, OGC Web Services Initiative Phase 2, address issues involving the adoption and integration of Web Services technology broadly adopted within the Information Technology industry by leading organizations such as W3C, OASIS, and the open-source community [15]. A major issue is about the question of whether and how to adopt the core Web Service technologies (WSDL, SOAP, and UDDI). Specifically, OWS-1.2 has executed two relevant experiments to address these issues, which are SOAP experiment and UDDI experiment.

3.4.1 OGC SOAP Experiment

The premise of OGC SOAP experiment (SOAP implementation of OGC) [16] is the belief that porting OWS services to Web Services will offer several key benefits, including:

- Distribution -- It will be easier to distribute geospatial data and applications across platforms, operating systems, computer languages, etc.
- Integration -- It will be easier for application developers to integrate geospatial functionality and data into their custom applications.

- Infrastructure -- The GIS industry could take advantage of the huge amount of infrastructure that is being built to enable the Web Services architecture -- including development tools, application servers, messaging protocols, security infrastructure, workflow definitions, etc.

This experiment discusses how OWS services can be ported to the SOAP-based Web Services and highlights various issues/problems that have been discovered. In this experiment, the appropriate XML Schema and Web Service Definition Language (WSDL) files were defined, which allow WMS services to be invoked using standard protocols such as HTTP GET, HTTP POST and SOAP. Standard Web Services toolkits, such as Visual Studio .NET, Apache Axis, XML Spy, and so on, were used to invoke OGC Web Services across the Internet.

A number of very specific issues on using Web Services standards and Web Service toolkits to call a WMS server have been identified. They are very specific to each adopted toolkit. For example, .NET does not support importing multiple XML Schema files that have the same target namespace. A workaround was found for each of the problems.

In summary, the adopted Web Service toolkits successfully invoked the services of a Web Map Server by successfully generating client proxies from a given WSDL. However, specific problems exist for different toolkits, and workaround solutions are available for most of the issues.

3.4.2 OGC UDDI Experiment

The OGC UDDI experiment [17] used UDDI (Universal Discovery, Description, and Integration) registries to discover geospatial content in general and OGC services in particular. This work was performed for the OGC's Interoperability Program OWS1.2 testbed initiative. Catalog interfaces and service information models have been developed within OGC specifically for geospatial purposes. This effort has been largely self-contained, however, and not particularly accessible from the general Web Services world. UDDI, on the other hand, has made the most progress of any service registry towards universal acceptance and accessibility, but has not been specifically adapted for geospatial applications.

The premise of the experiment is to determine whether and how the reach of UDDI might be combined with the geospatial focus of OGC services development to make geospatial content and services more universally discoverable and consumable by non-GIS users. The participants in this experiment will take a variety of approaches to coordinating OGC services and UDDI registries, as expressed in the User Scenarios below. The approaches all center around developing a crosswalk between the OGC and UDDI service information models.

The goal of the experiment will be to assess the practicality of both the crosswalk and the coordination scenarios, as well as to make concrete recommendations for improvements to either or both information models to further this purpose.

There were four general usage scenarios underlying this experiment. In Discover OGC Registries, user binds to a general purpose UDDI registry to discover specialized registries (and clients) for geospatial data and services. User then switches to Discover OGC Services, here user binds to a general-purpose UDDI registry to discover OGC services which have been published to it, either manually or automatically. In Discover OGC services with UDDI interface to OGC registry, user makes use of general purpose UDDI clients against OGC registries with UDDI interfaces to discover OGC services and build clients to them. In Publish OGC service to UDDI, user employs a general purpose UDDI publishing client to publish an OGC service directly to UDDI. The service metadata may or may not then be made available through a corresponding OGC registry interface or service.

The general design principle is that the OGC Service Information Model and the OGC Service Registry Model should be mapped onto the UDDI information model with as few changes as possible. Specifically, the UDDI registries for OGC services should be compatible with W3C standards efforts such as HTTP and XML, and compatible with other relevant OGC specification and pre-specifications efforts, including Service Information Model, General Service Model, Registry Information Model, and Web Catalog/Registry Service.

The UDDI experiment produced examples of discovering OGC services through UDDI interfaces, as well as means of mapping the UDDI metadata model to the metadata models used in OGC services. Experiments on spatial discovery and content discovery through UDDI showed that theses tasks are possible. However, the “fit” of the OGC models with the UDDI model and interfaces is poor. This is especially true when considering the capabilities of available UDDI clients to make full (or extended) use of the UDDI service interfaces. For example, testing with Sun UDDI registry demonstrates that this client lacks of resources for making use of the publisher API. The tentative conclusion drawn from this experiment is that UDDI Version 2 is best suited for discovering the existence of services based on very general taxonomic or classification criteria. It is not suited for obtaining the information to bind to a specific GIS service, and even less well suited to discovering specific contents or capabilities of individual GIS service instances.

3.4.3 Using WSDL in OGC Web Services

Even though, OGC does not have experiments to test the use of WSDL in OGC Web Services, there are discussions about the advantages and issues of adoption of WSDL into OGC Web Services.

An OpenGIS discussion paper has pointed out that OGC can take the following advantages of WSDL [19]:

- XML-schema based description provides references to abstract types from which service instances can attach instance specific service offer information
- Option for normative description of well known interfaces

- Syntax for instance specific “binding” of access point to “ports”.

On the other hand, the insufficiency of WSDL for GIS Web Services also has been recognized. WSDL was used in OWS 1.1 in the OperationSignatures portion of a service XML Capabilities document. WSDL was designed so that a programmer could discover and build an interface to a particular service instance, given an understanding of the semantics of a given operation. WSDL would allow automatic generation of “Stub” code for the interface. Basically a programmer would be required to extend this according to an understanding of the semantics of the content and the operations. Further knowledge of the content is required to “bind” to the service with a meaningful query.

The emphasis in WSDL definition is flexibility, inclusiveness, and extensibility. However, for the services in the GIS domain, the syntax of the operation is constant, and the content of service instances varies. The WSDL descriptions of “content access” services could be too general and abstract for GIS usage. WSDL does provide a syntactical framework for normative descriptions of “well known services” such as WMS.

3.5 An Example of Integration – ESRI ArcWeb

Besides the OGC experiments on integrating the general Web services technologies, some leading GIS software vendors have integrated the two areas in their implementation. The success of these projects is going to provide good evidence of the practicability of such an integration. ESRI ArcWeb is a good example of the GIS Web services, which integrates both core general technologies and OGC Web Services standards.

3.5.1 ArcWeb Overview

ESRI's ArcWeb Services [2, 3] are a type of Web service that provides spatial data and GIS functionality via the Internet to ArcGIS and custom Web applications. ArcWeb Services offer a way for developers to include GIS content and capabilities in their applications without having to host the data or develop GIS tools themselves, resulting in significant savings of development time, expense, and computer resources.

The services provided by ArcWeb are from simple mapping to complex tasks such as multipoint routing. The rich set of services makes creating lightweight, Web-enabled applications fast and simple. ArcWeb Services allow developers to use a rich offering of up-to-date data content, with more data being added all the time. The data accessible through ArcWeb include GDT United States Streets, USGS National Elevation Data, USGS National Land Cover Data, FEMA Q3 Flood Data, Census 2000 Population, ESRI Data & Maps, and so on [2].

3.5.2 ArcWeb Architecture

ArcWeb Services use SOAP to communicate so they are compatible with some of the latest Web service toolkits. Web service toolkits such as Microsoft .NET simplifies the implementation of ArcWeb Services because the communication protocol is handled automatically. ArcWeb Services can also communicate directly through SOAP for clients not using Web service toolkits. ArcWeb Services are described in WSDL. ArcWeb Services are published to a Universal Description, Discovery, and Integration (UDDI) registry so developers can easily discover them. The ArcWeb Architecture which integrates both the general Web Service technology and the OGC Web Services standards is shown in Figure 3-4 [2, 27].

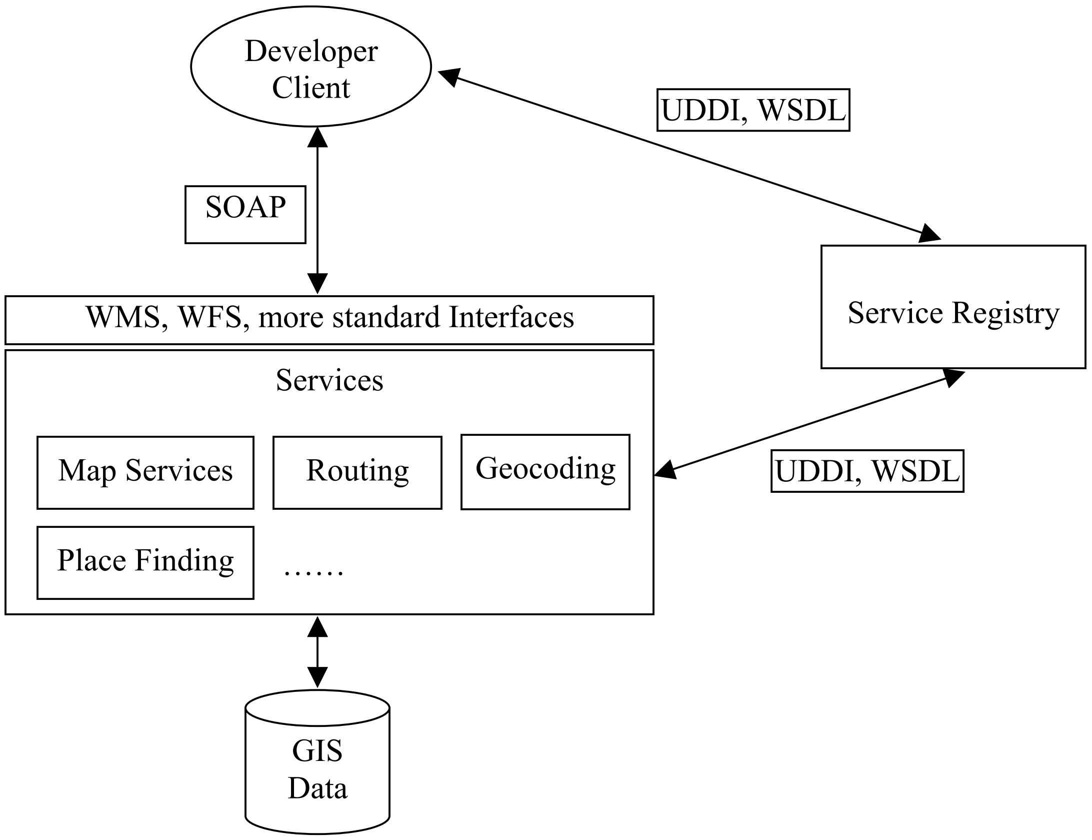


Figure 3-4. ArcWeb Architecture.

The key advantages of using ArcWeb for Developers are the following [2]:

- Access to vast amounts of current, reliable data and GIS capabilities without having to maintain or store the data or develop the GIS capabilities yourself;
- Ability to combine multiple services (such as address matching, routing, point of interest [POI] management, and more) and integrate these services with your own application environment, leading to limitless possibilities for sharing geographic information;
- No need to purchase hardware or software;
- No need to obtain updates to data sets because the data accessed via ArcWeb for Developers is always current;
- 24/7 reliability;
- Standards-based (SOAP/XML interface, Web Services Description Language [WSDL] access, published on the Universal Description, Discovery, and Integration [UDDI] registry).

CHAPTER 4 DESIGN AND IMPLEMENTATION: AN EXPERIMENT ON A SOAP-BASED GIS WEB APPLICATION

4.1 Overview

In this chapter, I report my experiment on a SOAP-based GIS Web application. This is an implementation of a client of Microsoft TerraService, which retrieves DOQ map images from the TerraService. It uses SOAP as the communication protocol. To demonstrate the use of map images, additional feature data from a local ArcSDE server are overlapped on the map images. Some related query functions are also implemented.

The Java Web Services Developer Pack (Java WSDP) 1.2 [25] is used to construct SOAP messages to communicate between the client and the TerraService. The TerraService Data Structures, such as AreaBoundingBox, AreaCoordinate, LonLatPt, …, are implemented in Java to provide parameters for the TerraService methods. SOAP messages are constructed using Java WSDP to communicate with TerraService methods such as GetAreaFromPt and GetTile.

4.2 Design Architecture

In this section, I introduce the design architecture of my implementation experiment. This implementation allows the user to provide the center longitude and latitude, width and height of the map. SOAP messages are constructed according the user input to retrieve DOQ images. ArcSDE feature data are then retrieved and displayed on the map image. Without storing any image data on the local disk, the user can get any map image area she/he wants from the TerraService. This again shows the beauty of Web Services. The design architecture of the implementation is depicted in Figure 4-1.

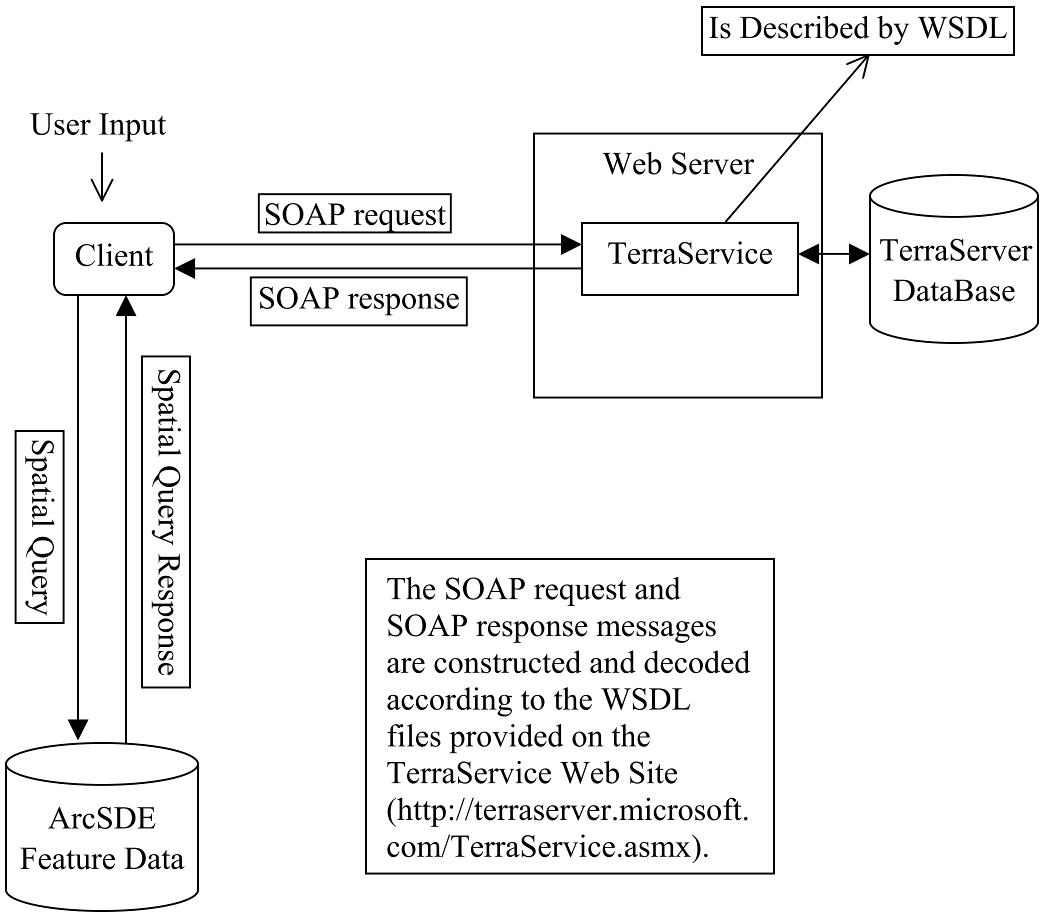

Is Described by WSDL
User Input
Web Server
SOAP request
Client
TerraService
TerraServer DataBase
SOAP response
Spatial Query
Spatial Query Response
ArcSDE Feature Data
The SOAP request and SOAP response messages are constructed and decoded according to the WSDL files provided on the TerraService Web Site (http://terraserver.microsoft.com/TerraService.asmx).

Figure 4-1. Implementation Design Architecture.

4.3 Implementation Workflow

The implementation is carried out according to the workflow shown in Figure 4-2. First, the program constructs SOAP request messages to get image tiles from TerraService according to the user input, which specifies the position (center longitude and latitude) and extent (width and height) of the map. Meanwhile, the meta data for the tiles are available in the SOAP response messages for latter use in creating the cropped map image. The cropping of the image is done after obtaining the images tiles and their meta data. The tile meta data also provide the input bounding box for the spatial query to retrieve ArcSDE feature data. Once the ArcSDE data are retrieved from the database, overlay of feature data and map image is done. Some query functions are implemented in the mean while.

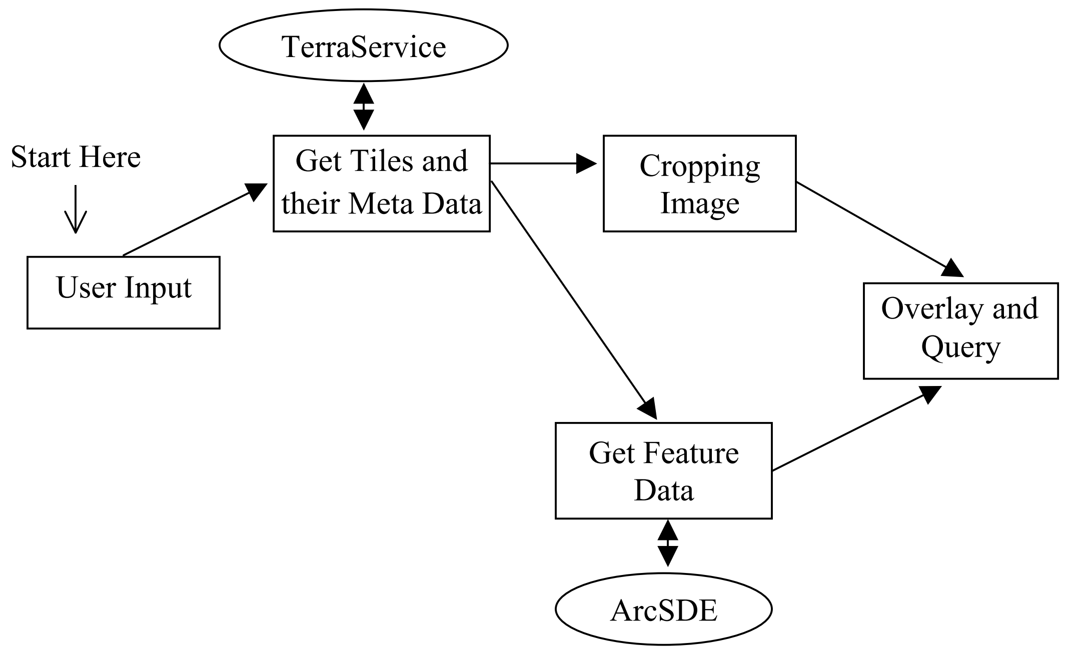


Figure 4-2. Workflow of Implementation Experiment

4.4 TerraServer Image Storage

Here I will provide some basic introduction of the image storage of TerraServer, which is necessary for understanding how the implementation works [5]. TerraServer supports a powers-of-2 ladder of resolutions - .25, .50, 1, 2, 4, 8, 16, 32, through 16,384 meters per pixel. The 1 meter per pixel resolution is used as the "base resolution" in the system. All other resolutions are a multiple of 2 from 1 meter resolution.

All TerraServer tiles are 200 x 200 pixel images. A 1 meter resolution tile covers a 200 meter by 200 meter area. A 2 meter tile covers a 400 meter by 400 meter area, and so on. Typically, TerraServer stores a seven-level "image pyramid". The actual number of resolutions is dependent on the type of imagery being stored. TerraServer stores several mosaic themes. Currently two themes are accessible from the TerraService -- aerial imagery (photograph), and topographic map (scanned from paper maps).

USGS Digital Ortho-Quadrangles (DOQ) images are gray-scale, 1-meter resolution aerial photos. Cars can be seen, but 1-meter resolution is too coarse to show people. Imagery is ortho-rectified to 1-meter square pixels. Approximately 90% of the U.S. has been digitized.

USGS Digital Raster Graphics (DRG) images are 13-color digitized topographic maps, with scales varying from 2.4 meter resolution to 25.6 meter resolution. DRGs are the digitized versions of the popular USGS topographic maps. A sample USGS DOQ image and USGS DRG image are shown in Figure 4-3 [5].

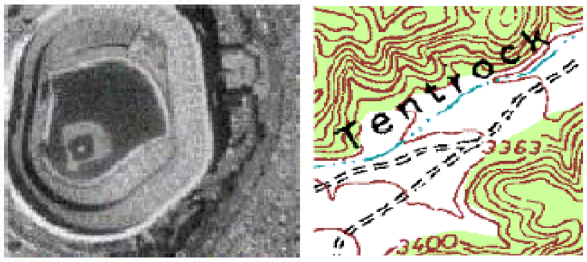


Figure 4-3. A USGS DOQ Image (Left) and A USGS DRG Image (Right) [5].

The most popular theme is the aerial imagery, which is the only image type used in this implementation.

4.5 Implementation Solution

4.5.1 Java Web Services Developer Pack

The Java Web Services Developer Pack (Java WSDP) is used as the implementation tool. Java WSDP is a free integrated toolkit that allows Java developers to build and test XML applications, Web services, and Web applications with the latest Web service technologies and standards implementations. Technologies in Java WSDP include the Java APIs for XML, Java Architecture for XML Binding (JAXB), JavaServer Faces, Web Services Interoperability Sample Application, XML Security, JavaServer Pages Standard Tag Library (JSTL), Java WSDP Registry Server, Ant Build Tool, and Apache Tomcat container.

Perhaps the most important feature of the JWSDP is that they all support industry standards, thus ensuring interoperability. Another feature of the JWSDP is that they allow a great deal of flexibility. Users have flexibility in how they use the APIs.

4.5.2 Construct SOAP Messages

Building an image from TerraServer Tiles involves "stitching together" several images tiles from TerraServer image database to create a new image cropped to some end user specified dimensions. The TerraService methods which are used to get map images in this project are *GetAreaFromPt* and *GetTile* (Table 4-1) [24].

GetAreaFromPt

```
public AreaBoundingBox GetAreaFromPt(LonLatPt center, Theme theme,
    Scale scale, int displayPixWidth, int displayPixHeight)
```

GetTile

```
public Byte[] GetTile(TileId id)
```

Table 4-1. TerraService Methods GeAreaFromPt and GetTile.

The *GetAreaFromPt* method returns the tile meta-data for a Geographic rectangle. Use this call to identify the tiles required to construct an image of a specific size and resolution with a known center point. The *GetAreaFromPt* is typically called in applications that want to control the size of the display area. Using *GetAreaFromPt*, the caller controls the specific Longitude and Latitude point to be displayed in the center. The LonLatPt point parameter identifies the Geographic center of the rectangle of interest. The Theme and Scale input parameters identify the type imagery and resolution of interest. The *displayPixWidth* and *displayPixHeight* parameters identify the size of image the caller intends to create. The GetAreaFromPt method computes the TerraServer tiles that overlap the corners of your image area in the resolution specified in the Scale parameter.

The GetTile method returns a Byte array containing the compressed image data for the requested tile. The TileId input parameter identifies the specific data row in the TerraServer database to be returned.

The first step is to construct a request SOAP message to call the *GetAreaFromPt* method to identify the TerraServer image tiles that are required to contribute to the user required image.

The request SOAP message is shown in Table 4-2 [24]:

```
    POST /TerraService.asmx HTTP/1.1
    Host: terraserver-usa.com
    Content-Type: text/xml; charset=utf-8
    Content-Length: length
    SOAPAction: "http://terraserver-usa.com/terraserver/GetAreaFromPt"

    <?xml version="1.0" encoding="utf-8"?>
    <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
      <soap:Body>
        <GetAreaFromPt xmlns="http://terraserver-usa.com/terraserver/">
          <center>
            <Lon>double</Lon>
            <Lat>double</Lat>
          </center>
          <theme>Photo or Topo or Relief</theme>
          <scale>Scale1mm or Scale2mm or Scale4mm or Scale8mm or
    Scale16mm or Scale32mm or Scale63mm or Scale125mm or Scale250mm or
    Scale500mm or Scale1m or Scale2m or Scale4m or Scale8m or Scale16m
    or Scale32m or Scale64m or Scale128m or Scale256m or Scale512m or
    Scale1km or Scale2km or Scale4km or Scale8km or Scale16km</scale>
          <displayPixWidth>int</displayPixWidth>
          <displayPixHeight>int</displayPixHeight>
        </GetAreaFromPt>
      </soap:Body>
</soap:Envelope>
```

Table 4-2. GetAreaFromPt SOAP Request Message.

Table 4-3 shows the Java code to construct the GetAreaFromPt SOAP request message.

```
      String lon; // center longitude of the requested image
      String lat; // center latitude of the requested image
      String them; // theme of the requested image
      String scal; // scale of the requested image
      String Width; // width of the requested image
      String Height; // height of the requested image

      // Create the SOAP message
      MessageFactory factory =MessageFactory.newInstance();

      SOAPMessage message = factory.createMessage();
      SOAPPart soapPart = message.getSOAPPart();
      SOAPEnvelope envelope = soapPart.getEnvelope();
      SOAPHeader header = envelope.getHeader();
      header.detachNode();

envelope.addNamespaceDeclaration("xsi","http://www.w3.org/2001/XMLSchema-
instance");
envelope.addNamespaceDeclaration("xsd","http://www.w3.org/2001/XMLSchema");
envelope.addNamespaceDeclaration("soap",
"http://schemas.xmlsoap.org/soap/envelope/");
      SOAPBody body = envelope.getBody();

      SOAPElement getAreaFromPt =body.addChildElement("GetAreaFromPt");
      getAreaFromPt.addNamespaceDeclaration("","http://terraserver-
usa.com/terraserver/");
      SOAPElement center = getAreaFromPt.addChildElement("center");
      SOAPElement Lon = center.addChildElement("Lon");
      Lon.addTextNode(lon);
      SOAPElement Lat = center.addChildElement("Lat");
      Lat.addTextNode(lat);
      SOAPElement theme = getAreaFromPt.addChildElement("theme");
      theme.addTextNode(them);
      SOAPElement scale = getAreaFromPt.addChildElement("scale");
      scale.addTextNode(scal);
      SOAPElement pixWidth =
getAreaFromPt.addChildElement("displayPixWidth");
      pixWidth.addTextNode(Width);
      SOAPElement pixHeight =
getAreaFromPt.addChildElement("displayPixHeight");
      pixHeight.addTextNode(Height);
```

Table 4-3. Code for Constructing the SOAP Request.

After getting connected to TerraService and sending over the request SOAP message, the response is sent back from the TerraService, which looks like the SOAP message shown in Table 4-4 [24]:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetAreaFromPtResponse xmlns="http://terraserver-
usa.com/terraserver/">
      <GetAreaFromPtResult>
        <NorthWest>
          <TileMeta>
            <Id xsi:nil="true" />
            <TileExists>boolean</TileExists>
            <NorthWest xsi:nil="true" />
            <NorthEast xsi:nil="true" />
            <SouthWest xsi:nil="true" />
            <SouthEast xsi:nil="true" />
            <Center xsi:nil="true" />
            <Capture>dateTime</Capture>
          </TileMeta>
          <Offset>
            <Point xsi:nil="true" />
            <XOffset>int</XOffset>
            <YOffset>int</YOffset>
          </Offset>
        </NorthWest>
        <NorthEast>
            ...
        </NorthEast>
        <SouthWest>
            ...
        </SouthWest>
        <SouthEast>
            ...
        </SouthEast>
        <Center>
          ...
        </Center>
        <NearestPlace>string</NearestPlace>
        <OverlappingThemeInfos>
          <OverlappingThemeInfo>
            <LocalTheme>boolean</LocalTheme>
            <Theme>Photo or Topo or Relief</Theme>
            <Point xsi:nil="true" />
            <ThemeName>string</ThemeName>
            <Capture>dateTime</Capture>
            <ProjectionId>Geographic or UtmNad27 or
UtmNad83</ProjectionId>
            <LoScale>Scale1mm or Scale2mm or …… or
Scale16km</LoScale>
            <HiScale>Scale1mm or Scale2mm or …… or
Scale16km</HiScale>
            <Url>string</Url>
          </OverlappingThemeInfo>
          <OverlappingThemeInfo>
            <LocalTheme>boolean</LocalTheme>
            <Theme>Photo or Topo or Relief</Theme>
            <Point xsi:nil="true" />
            <ThemeName>string</ThemeName>
            <Capture>dateTime</Capture>
            <ProjectionId>Geographic or UtmNad27 or
UtmNad83</ProjectionId>
            <LoScale>Scale1mm or Scale2mm or …… or
Scale16km</LoScale>
            <HiScale>Scale1mm or Scale2mm or …… or
Scale16km</HiScale>
            <Url>string</Url>
          </OverlappingThemeInfo>
        </OverlappingThemeInfos>
      </GetAreaFromPtResult>
    </GetAreaFromPtResponse>
  </soap:Body>
</soap:Envelope>
```

Table 4-4. GetAreaFromPt SOAP Response Message.

The Java code for sending the GetAreaFromPt SOAP request message and receiving the GetAreaFromPt SOAP response message is shown in Table 4-5.

```
      byte[] b; // convert the GetAreaFromPt SOAP request message into
      // byte array

      // Create the connetion
      URL endpoint = new URL("http://terraserver-
usa.com/terraservice.asmx");
      HttpURLConnection connection =(HttpURLConnection)
endpoint.openConnection();

      connection.addRequestProperty("Content-Type", "text/xml;
charset=utf-8");
      connection.addRequestProperty("Content-
Length",Integer.toString(b.length));
      connection.addRequestProperty("SOAPAction",
"\"http://terraserver-usa.com/terraserver/GetAreaFromPt\"");
      connection.setRequestMethod("POST");
      connection.setDoOutput(true);
      connection.setDoInput(true);

      // Send out the request
      OutputStream out = connection.getOutputStream();
      out.write(b);
      out.flush();
      out.close();
      System.out.println(connection.getResponseMessage());

      // Get the response
      MessageFactory factory = MessageFactory.newInstance();
      MimeHeaders mm = new MimeHeaders();
      mm.addHeader("Content-Type","text/xml");
      SOAPMessage response =
factory.createMessage(mm,connection.getInputStream());

      // Get the content of the response
      SOAPPart sp = response.getSOAPPart();
      SOAPEnvelope se = sp.getEnvelope();
      SOAPBody sb = se.getBody();
      Name bodyName =
se.createName("GetAreaFromPtResponse","","http://terraserver-
usa.com/terraserver/");
      Name elementName = se.createName("GetAreaFromPtResult");
      Iterator it = sb.getChildElements(bodyName);
      SOAPBodyElement bodyElement =(SOAPBodyElement)it.next();
      Iterator it2 = bodyElement.getChildElements();
      SOAPElement getAreaResult = (SOAPElement)it2.next();
      Iterator it3 = getAreaResult.getChildElements();
      SOAPElement nwest = (SOAPElement)it3.next();
      SOAPElement neast = (SOAPElement)it3.next();
      SOAPElement swest = (SOAPElement)it3.next();
      SOAPElement seast = (SOAPElement)it3.next();
      SOAPElement center = (SOAPElement)it3.next();
               ...
```

Table 4-5. Code for Sending the SOAP Request and Receiving the SOAP Response.

The GetAreaFromPt SOAP response message provides all the information needed to get the required tiles and the meta data of the tiles to do the image cropping. Similar SOAP messages were constructed to get each image tile. All the SOAP request and response messages, including the two messages shown in Table 4-2 and Table 4-4, for using the TerraService are available at TerraService WSDL web page [24].

4.5.3 Get Terraserver Image Tiles and Create the Cropped Map Image

The response SOAP message also contains the meta-data describing a single image tile in the TerraServer database, such as **TileId**, which identifies a unique image tile in the TerraServer database, and the **LonLatPtOffset**, which describes the pixel location of a specific longitude and latitude value within a TerraServer image tile. Here the Offsets identifie the 4 corner points in the NorthWest, NorthEast, SouthWest, SouthEast tiles and the center point in the Center tile. After obtaining the TileIds, the program calls *GetTile* to get a **BufferedImage** of each tile. Then according to the Offsets, the image cropping is done following for loop (Table 4-6).

```
      int x;  // current Tile row id
      int y;  // current Tile column id
      int xstart; // row id of the NorthWest tile
      int ystart; // column id of the NorthWest tile
      TileId SWId; // TileId of the SouthWest tile
      TileId NEId; // TileId of the NorthEast tile
      GetTile gt;  // GetTile constructs SOAP message to call
      // TerraService GetTile method and build a BufferedImage of that
      // tile
      int imgX; // offset x
      int imgY; // offset y
      TileId tid; // the current tile id
      Vector brv; // a vector stores the cropped map image tiles

      // SWId.Y is the column id of SouthWest tile
      for (int y = ystart; y >= SWId.Y; y--) {
            // NEId.X is the row id of NorthEast tile
            for (int x = xstart; x <= NEId.X; x++) {
                        ...
                  BufferedImage imb = gt.getImage(tid.Theme.themebis,
                  tid.Scale.scalebis , tid.Scenebis , tid.Xbis ,
                  tid.Ybis);
                        ...
                  imb = imb.getSubimage(imgX, imgY, imgWidth,
                  imgHeight);
                        ...
                  brv.add(imb);
                        ...
            }
}
```

Table 4-6. Code for Image Cropping.

The number of rows (*R*) and number of columns (*C*) of the cropped tiles need to be remembered for the later displaying purpose. To display the cropped image, the program retrieves each tile from the image tile vector in a flow layout order, and displays them in an *R* by *C* matrix.

Now a cropped image from a set of TerraServer image tiles has been produced. Note, when receiving the meta data of the tiles, the NWOffset (Offset of the NorthWest tile) and SEOffset (Offset of the SouthEast tile) also need to be remembered to provide the bounding box to retrieve the ArcSDE feature data for this map area, and to calculate the position of a latitude-longitude point on the map and draw the feature data on the map.

4.5.4 Visualizing Map Image Cropping

Figure 4-4 demonstrates how the desired map image is cropped from the TerraServer tiles. The dashed rectangle represents the user requested map image area. The point *p5* is the center (Offset of the center tile) of the map. The other four points *p1*, *p2*, *p3*, and *p4* represents the NortheWest, NorthEast, SouthWest, and SouthEast corners (tile Offsets) of the map respectively.

TerraServer image tile *T1*, *T2*, *T3*, ..., *T9* are the 9 tiles required to build the desired map image. Among the 9 tiles, the tile-meta information of 5 of them is available for doing the image cropping, and they are: *T1* -- NorthWest tile, *T3* -- NorthEast tile, *T7* -- SouthWest tile, *T9* -- SouthEast tile, *T5* -- Center tile.

Provided with the image tiles from TerraServer and the Offsets of the 4 corner tiles, you can get the user requested map image.

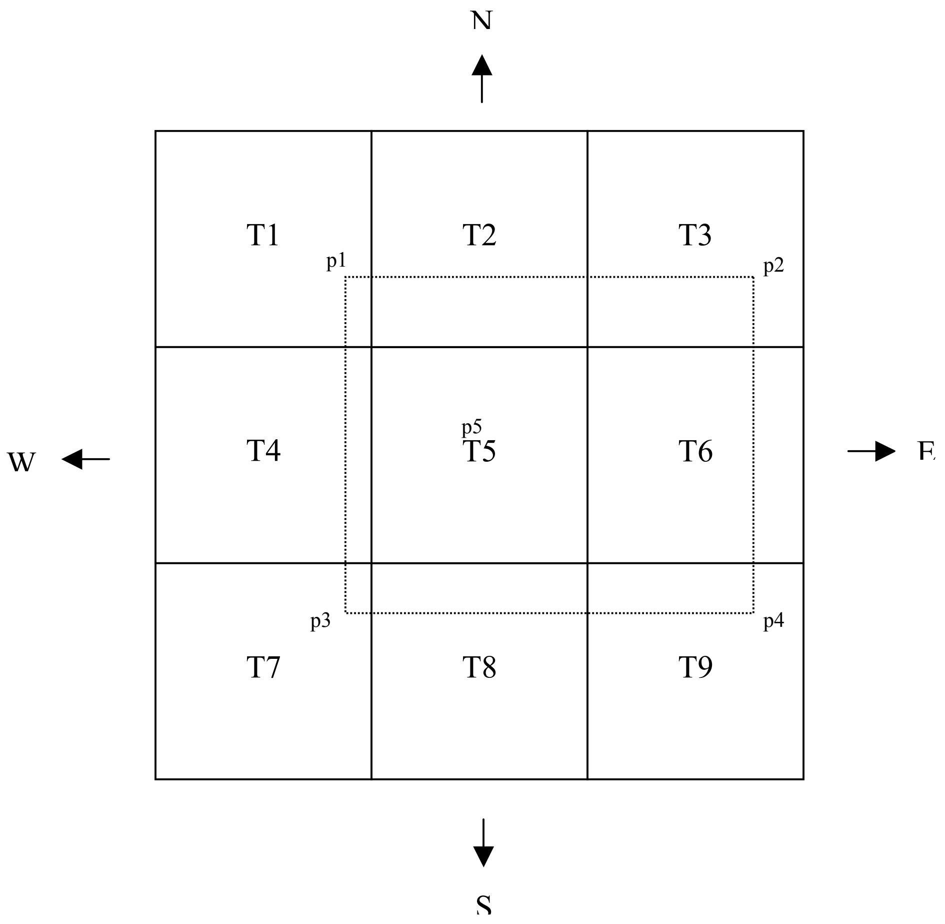


Figure 4-4. Visualizing Map Image Cropping

4.5.5 Zoom Out

The map images with higher resolution (higher than 1m, such as 500mm, 250mm, and 125mm) are not available from the TerraService. The program zooms out the map image with 1m resolution to get the higher resolutions. Let *scalefactor* = 1000(mm)/current resolution(mm), and draw the corresponding 1m resolution BufferedImage in a *scalefactor* times bigger area (Table 4-7).

```
Graphics g;
BufferedImage image;
      ...
g.drawImage(image, 0, 0, image.getWidth() * scalefactor,
image.getHeight()*scalefactor, Color.white, null);
```

Table 4-7. Code for Zoom Out.

4.5.6 Retrieve ArcSDE Feature Data

The GetMoritorData class takes the bounding box as the input parameters, and retrieves the ArcSDE feature data (point data here) within this bounding box (Table 4-8).

```
public GetMoritorData(double minX , double minY , double maxX ,
double maxY ){
      SpatialQueryEx sp=new SpatialQueryEx("moritor");
      Vector vector=sp.queryInBox(minX, minY, maxX, maxY);
            ...
}
```

Table 4-8. Code for Retrieving ArcSDE Feature Data.

So provided with the bounding box remembered from GetAreaFromPt SOAP response, the ArcSDE feature data from the database for the map area are easily retrieved (Table 4-9).

```
// the following data provides the bounding box information
// remembered from Step 4.
LonLatPtOffset nwoffset; // Offset of NorthWest tile
LonLatPtOffset seoffset;  // Offset of SouthEast tile
LonLatPt nwpt = nwoffset.Point; // LonLatPt  of nwoffset
LonLatPt sept = seoffset.Point; // LonLatPt of seoffset

GetMoritorData gmd = new GetMoritorData(nwpt.Lon, sept.Lat,
sept.Lon, nwpt.Lat);
```

Table 4-9. Obtain Bounding Box for ArcSDE Feature Data.

Spatial query is performed to retrieve the feature data. The spatial query is executed in the method runSpatialQuery, which is called in queryInBox. The essential code of queryInBox and runSpatialQuery is show in the Table 4-10.

```
public Vector queryInBox(double minX , double minY , double maxX
, double maxY ) {
  Vector vector=new Vector();
  // Generate a rectangular shape that will be used as a
  // filter
  try{
    SeShape shape = new SeShape(layer.getCoordRef());
    SeExtent extent = new SeExtent( minX,minY,maxX,maxY);
    shape.generateRectangle(extent);
    SeShape[] shapes = new SeShape[1];
    shapes[0] = shape;
```

```
     // Retrieve all the shapes that are contained within the
     // rectangles envelope.
         ...
     vector=runSpatialQuery(shapes, SeFilter.METHOD_ENVP );
         ...
  }catch( SeException e) {}
    return vector;
  } // End method queryInBox

  // Runs a spatial query against the layer using the
  // specified shape and method

 public Vector runSpatialQuery( SeShape[] shape, int method) {
     Vector vector;
        ...
    try {
        ...
      SeQuery spatialQuery = null;
      SeSqlConstruct sqlCons = new SeSqlConstruct(
      layer.getName() );
      // conn is the ArcSDE connection, cols is a String array
      // containing table column definitions
      spatialQuery = new SeQuery(conn, cols, sqlCons);
      spatialQuery.prepareQuery();

      // Set spatial constraints
         ...
      spatialQuery.execute();
      SeRow row = spatialQuery.fetch();
         ...
      spatialQuery.close();
   } catch ( SeException sexp ) {}
      return vector;
   } // End method runSpatialQuery
```

Table 4-10. Spatial Query for Retrieving ArcSDE Feature Data.

4.5.7 Overlay Feature Data with Map Image

Provided with the width and height of the map image and the latitude and longitude of the NorthWest and SouthEast corners of the map, the latitude and longitude of each pixel of the map can be calculated with great accuracy. Similarly, each latitude-longitude point data can be positioned on the map accurately. The latitude and longitude measurement of the NorthWest and SouthEast corners are remembered from the GetAreaFroPt SOAP response. The code in Table 4-11 draws a point data on the map image.

```
LonLatPt temp; // the point data
double lon = temp.Lon; // longitude of the point
double lat = temp.Lat;  // latitude of the point
int x; // x position of the point on the map
int y;  // y position of the point on the map
Graphics g;

// calculate the x position of the feature point data
x = (int)(((lon - nwpt.Lon)*10000000)/((sept.Lon - nwpt.Lon) *
10000000/(this.width * scalefactor))) + xstart;

// calculate the y position of the feature point data
y = (int)(((lat - nwpt.Lat)*10000000)/((sept.Lat - nwpt.Lat) *
10000000/(this.height * scalefactor))) + ystart;

// draw the feature point data on the map
g.fillRect(x - 1, y - 1, 2, 2);
```

Table 4-11. Code for Overlaying Feature Data and Map Image.

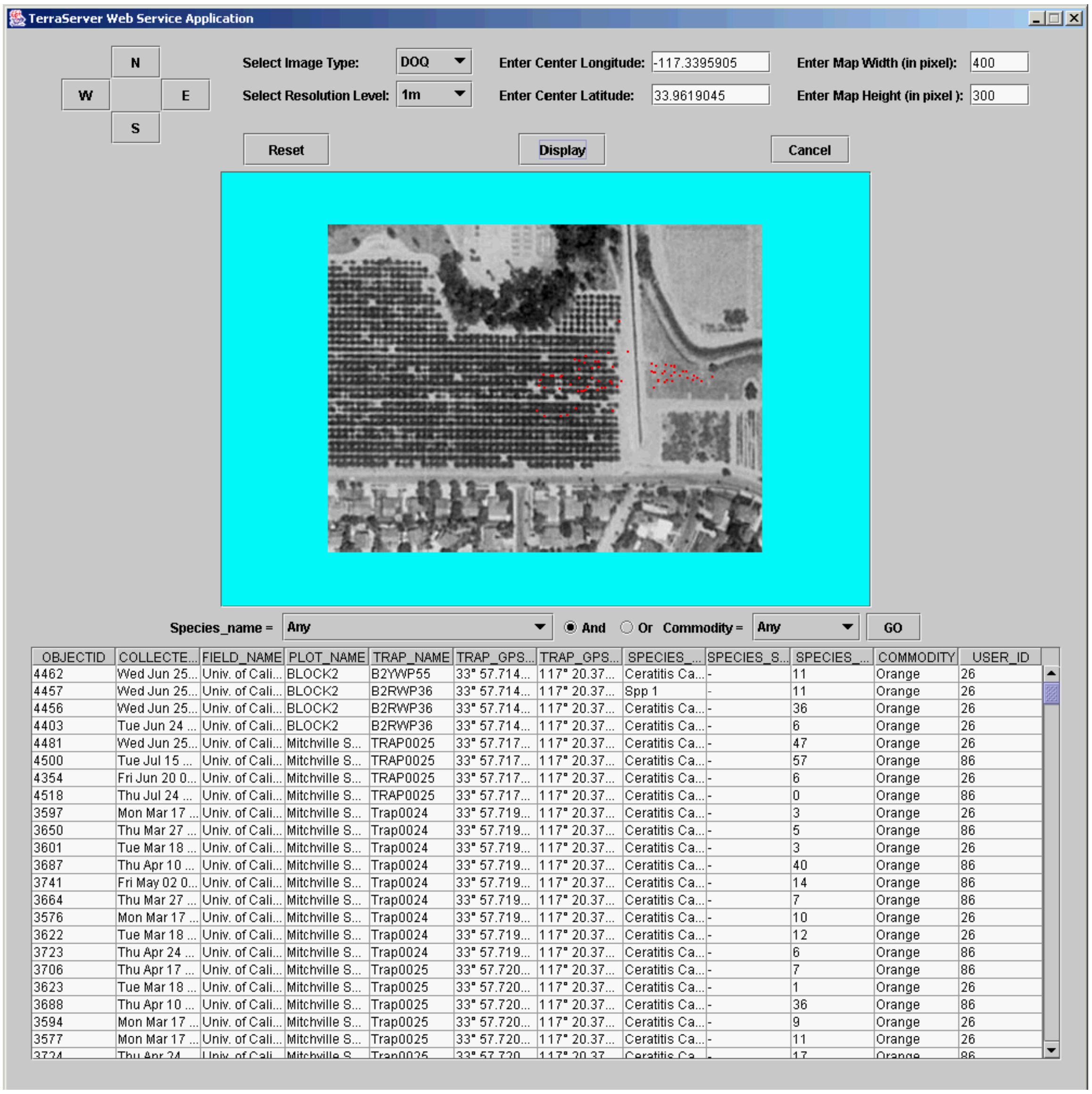
In the same manner, once the program draws the point, it can find and highlight it on the map provided with its latitude and longitude. This is how the query feature data part is done.

4.5.8 Summary of Implemented Functionality

In summary, a good set of GIS functionalities have been implemented in this case study, which includes:

- Dynamically retrieves the up-to-date image data from the TerraService, provided with the center point (longitude-latitude) and width and height of the map;
- Moving the map image to any of the four directions, East, West, North, and South;
- Retrieves feature data from ArcSDE;
- Overlay the point feature data with the map image;
- Query the feature data, and display the query on the map.

The Java WSDP technology makes the implementation easy, flexible, and extensible. A sample run of the implemented program is shown in Figure 4-5.



Select Image Type: DOQ
Select Resolution Level: 1m
Enter Center Longitude: -117.3395905
Enter Center Latitude: 33.9619045
Enter Map Width (in pixel): 400
Enter Map Height (in pixel): 300

Species_name = Any [✓ And] Or Commodity = Any

OBJECTID	COLLECTE...	FIELD_NAME	PLOT_NAME	TRAP_NAME	TRAP_GPS...	TRAP_GPS...	SPECIES_...	SPECIES_S...	SPECIES_...	COMMODITY	USER_ID
4462	Wed Jun 25...	Univ. of Cali...	BLOCK2	B2YWP55	33° 57.714...	117° 20.37...	Ceratitis Ca...	-	11	Orange	26
4457	Wed Jun 25...	Univ. of Cali...	BLOCK2	B2RWP36	33° 57.714...	117° 20.37...	Spp 1	-	11	Orange	26
4456	Wed Jun 25...	Univ. of Cali...	BLOCK2	B2RWP36	33° 57.714...	117° 20.37...	Ceratitis Ca...	-	36	Orange	26
4403	Tue Jun 24 ...	Univ. of Cali...	BLOCK2	B2RWP36	33° 57.714...	117° 20.37...	Ceratitis Ca...	-	6	Orange	26
4481	Wed Jun 25...	Univ. of Cali...	Mitchville S...	TRAP0025	33° 57.717...	117° 20.37...	Ceratitis Ca...	-	47	Orange	26
4500	Tue Jul 15 ...	Univ. of Cali...	Mitchville S...	TRAP0025	33° 57.717...	117° 20.37...	Ceratitis Ca...	-	57	Orange	86
4354	Fri Jun 20 0...	Univ. of Cali...	Mitchville S...	TRAP0025	33° 57.717...	117° 20.37...	Ceratitis Ca...	-	6	Orange	26
4518	Thu Jul 24 ...	Univ. of Cali...	Mitchville S...	TRAP0025	33° 57.717...	117° 20.37...	Ceratitis Ca...	-	0	Orange	86
3597	Mon Mar 17 ...	Univ. of Cali...	Mitchville S...	Trap0024	33° 57.719...	117° 20.37...	Ceratitis Ca...	-	3	Orange	26
3650	Thu Mar 27 ...	Univ. of Cali...	Mitchville S...	Trap0024	33° 57.719...	117° 20.37...	Ceratitis Ca...	-	5	Orange	86
3601	Tue Mar 18 ...	Univ. of Cali...	Mitchville S...	Trap0024	33° 57.719...	117° 20.37...	Ceratitis Ca...	-	3	Orange	26
3687	Thu Apr 10 ...	Univ. of Cali...	Mitchville S...	Trap0024	33° 57.719...	117° 20.37...	Ceratitis Ca...	-	40	Orange	86
3741	Fri May 02 0...	Univ. of Cali...	Mitchville S...	Trap0024	33° 57.719...	117° 20.37...	Ceratitis Ca...	-	14	Orange	86
3664	Thu Mar 27 ...	Univ. of Cali...	Mitchville S...	Trap0024	33° 57.719...	117° 20.37...	Ceratitis Ca...	-	7	Orange	86
3576	Mon Mar 17 ...	Univ. of Cali...	Mitchville S...	Trap0024	33° 57.719...	117° 20.37...	Ceratitis Ca...	-	10	Orange	26
3622	Tue Mar 18 ...	Univ. of Cali...	Mitchville S...	Trap0024	33° 57.719...	117° 20.37...	Ceratitis Ca...	-	12	Orange	26
3723	Thu Apr 24 ...	Univ. of Cali...	Mitchville S...	Trap0024	33° 57.719...	117° 20.37...	Ceratitis Ca...	-	6	Orange	86
3706	Thu Apr 17 ...	Univ. of Cali...	Mitchville S...	Trap0025	33° 57.720...	117° 20.37...	Ceratitis Ca...	-	7	Orange	86
3623	Tue Mar 18 ...	Univ. of Cali...	Mitchville S...	Trap0025	33° 57.720...	117° 20.37...	Ceratitis Ca...	-	1	Orange	26
3688	Thu Apr 10 ...	Univ. of Cali...	Mitchville S...	Trap0025	33° 57.720...	117° 20.37...	Ceratitis Ca...	-	36	Orange	86
3594	Mon Mar 17 ...	Univ. of Cali...	Mitchville S...	Trap0025	33° 57.720...	117° 20.37...	Ceratitis Ca...	-	9	Orange	26
3577	Mon Mar 17 ...	Univ. of Cali...	Mitchville S...	Trap0025	33° 57.720...	117° 20.37...	Ceratitis Ca...	-	11	Orange	26
3724	Thu Apr 24	Univ. of Cali	Mitchville S	Trap0025	33° 57.720	117° 20.37	Ceratitis Ca	-	17	Orange	86

Figure 4-5. A Sample Run of the Implementation.

CHAPTER 5 CONCLUSIONS

In this thesis project, I have reviewed the current status of integrating the core SOAP-based Web Services technologies and the OGC Web Services standards. The benefits to the distributed GIS computing by bringing these two areas together are emphasized. Basically, the integration will make it easier to distribute geospatial data and applications across platforms, operating systems, computer languages, etc; and it will make the GIS industry take advantage of the huge amount of powerful technologies that are being built to enable the general Web Services architecture.

OGC has started making efforts porting the SOAP-based Web Service technologies into their interoperability programs. However, these completed activities only stressed a small range of issues based on some specific specifications. The current integration status is still about whether and how to integrate. There are no conclusive answers to the whether question yet, and even farther from answering the how question. Obviously, more experiments are needed to reach a more conclusive answer of the integration questions.

The leading GIS software implementation incorporating both areas, like ESRI ArcWeb, has provided good evidence of the feasibility of integration. In my own case study, a good set of GIS functionalities has been implemented using the TerraService, which also involves both areas.

Besides the integration issue, the SOAP-based Web Services technology itself is still in the development process in many aspects, e.g., security, reliability, service quality, and services interactions. These aspects are also concerns of OGC Web Services. For a better future integration, the early consideration of Geographic computing characteristics in developing the general Web Services technology is a not a bad idea.

REFERENCES

1. Nadine Alameh, Chaining Geographic Information Web Services. IEEE Internet Computing, September/ October, 2003 (Vol. 7, No. 5).
2. An Overview of ArcWeb for Developers. ESRI white paper 2003. http://support.esri.com/index.cfm?fa=knowledgebase.whitepapers.listPapers&PID=20.
3. An Overview of ArcWeb Services. ESRI white paper 2003. http://support.esri.com/index.cfm?fa=knowledgebase.whitepapers.listPapers&PID=20.
4. ArcSDE Developer Help 8.3, http://arcsdeonline.esri.com/index.htm.
5. Tom Barclay, Jim Gray, Eric Strand, Steve Ekblad, Jeffrey Richter, 2002, TerraService.NET: An Introduction to Web Services. http://research.microsoft.com/research/pubs/view.aspx?msr_tr_id=MSR-TR-2002-53.
6. Christopher A. Brooks, An introduction to Web Services. http://www.cs.usask.ca/grads/cab938/An%20Introduction%20to%20Web%20Services.pdf.
7. Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI.

8. Allan Doyle, Carl Reed, Jeff Harrison, Mark Reichardt, Introduction to OGC Web Services, OGC Interoperability Program White Paper. http://ip.opengis.org/ows/010526_OWSWhitepaper.doc.

9. Geography Markup Language, OpenGIS Specifications, http://www.opengis.org/docs/02-023r4.pdf.

10. K. Gottschalk, S. Graham, H. Kreger, and J. Snell, Introduction to Web Services Architecture. http://www.research.ibm.com/journal/sj/412/gottschalk.html.

11. Steve Graham, Simeon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, Ryo Neyama, Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, 2002. Sams Publishing, Indianapolis, Indiana.

12. Louis Hecht Jr., Web Services Are the Future of Geoprocessing. http://www.geoplace.com/gw/2002/0206/0206opng.asp.

13. Kenji Hiraishi, The Summary of OGC Activities, http://www.acrors.ait.ac.th/digital_asia/GMSpapers/A91_KENJI_HIRAISHI.pdf.

14. Heather Kreger, Web Services Conceptual Architecture (WSCA 1.0), May 2001. http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf.

15. OGC Request for Technology, In Support of an OGC1 Web Services Initiative. April 2003. http://www.opengis.org/press/?page=pressrelease&view=20030430_OWS2_RFT_PR.

16. OGC Web Services SOAP Experiment Report, OpenGIS Specifications, http://www.opengis.org/docs/03-014.pdf.

17. OGC Web Services UDDI Experiment, OpenGIS Specifications, http://www.opengis.org/docs/03-028.pdf.

18. OpenGIS Consortium, http://www.opengis.org/specs/?page=specs.

19. OpenGIS Web Services Architecture, OpenGIS Discussion Paper, 2003. http://www.opengis.org/docs/03-025.pdf.

20. Zhong-Ren Peng and Ming-Hsiang Tsou, Internet GIS: Distributed Geographic Information Services for the Internet and Wireless Networks, 2003. John Wiley & Sons, Inc., Hoboken, New Jersey.

21. SOAP Attachment, http://www.w3.org/TR/SOAP-attachments.

22. SOAP Part 1, http://www.w3.org/TR/soap12-part1/.

23. SOAP Part 2, http://www.w3.org/TR/soap12-part2/.

24. TerraServer, http://terraserver-usa.com/.

25. The Java Web Services Tutorial, http://java.sun.com/webservices/docs/1.1/tutorial/doc/.

26. The Universal Description, Discovery and Integration (UDDI) protocol, http://www.uddi.org/.

27. Understanding ArcWeb Services for Developers: An Overview to SOAP Implementation. ESRI white paper 2003. http://support.esri.com/index.cfm?fa=knowledgebase.whitepapers.listPapers&PID=20.

28. Web Coverage Service, OpenGIS Specifications, http://www.opengis.org/docs/03-065r6.pdf.

29. Web Feature Services, OpenGIS Specifications, http://www.opengis.org/docs/02-058.pdf.

30. Web Map Service, OpenGIS Specifications, http://www.opengis.org/docs/03-086.pdf.

31. Web Services Architecture, http://www.w3.org/TR/2003/WD-ws-arch-20030808/.

32. Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl.

33. Web Services Interoperability, http://www.ws-i.org/.

VITA

Shujing Shu was born in Tianjin, China in 1972. She earned a Bachelor Degree and a Master Degree in Psychology during her study at Peking University, China from 1990 to 1998. She was a University of Iowa Fellowship and conducted research on Animal Cognition between September 1998 and December 2000.

She was enrolled in the graduate program in Computer Science Department at the University of New Orleans in January 2001. She worked on a research project in the Geography Department as a software developer from July 2002 till her graduation. She has great interest and rich experience in developing GIS and Web Services applications.