An Implementation of Hierarchical Inter-domain Routing in the context of UCLPv2

Master Thesis By

Qi Wang

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

Ottawa-Carleton Institute for Computer Science
School of Information Technology and Engineering
University Of Ottawa
Ottawa, Ontario, Canada

January 15, 2007

Abstract

As customer-owned and managed optical networks gain more popularity with large enterprises and institutions, the need to enable end-users to dynamically provision and configure network resources inspires the development of the second version of User Controlled Lightpath Provisioning System (UCLPv2).

The UCLPv2 software is based on a service-oriented architecture in which network resources are exposed as and managed through Web services. Furthermore, it introduces a new concept called Articulated Private Network, which is a collection of resources that end-users can change or articulate its topology dynamically.

The current system lacks the capability to automatically establish an end-to-end connection given a pair of source and destination switches. The thesis describes the new intra-domain and inter-domain routing functionalities as an enhancement to the UCLPv2. Furthermore, it focuses on the design and implementation of the inter-domain routing, which is based on the hierarchical management architecture for networks with condo-switches. The routing algorithm is implemented through the adaptation of the Dijkstar's algorithm to a hypergraph.

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Gregor von Bochmann for his insightful feedback and motivational guidance. Over the past two years, he has taught me how to engage in research activities and write an academic paper. I am also grateful to Dr. Michel Savoie, Scott Campbell, Hanxi Zhang, who are the UCLPv2 development team members at Communication Research Center, for their cooperation and support.

Table of Contents

List of Figures

Chapter 1 - Introduction

In recent years, the idea of customer-owned and managed optical network becomes popular. This type of network is rooted in the metro dark fiber network and long-haul wavelength network. Both research and commercial organizations have purchased or leased dark fibers or wavelengths to acquire the right of usage of these resources. Furthermore, these organizations can sublease the resources to their customers. A common practice is that organizations share the capital cost of network deployment. For example, they together purchase an optical switch, and each organization acquires ownership of a subset of ports on the switch. Similarly, organizations can share the cost of purchasing a single fiber, and each acquires ownership of a subset of channels on the fiber. Because organizations share network resources in a condominium fashion, this type of network is often referred as condominium network. Similarly, because a switch within the condominium network can be shared by different organizations, such type of switch is known as condo switch. An organization owning a port of the switch also owns the fiber attached to the port [1].

A system for User Controlled Lightpath Provisioning (UCLPv2) [2] was developed to facilitate the management of customer-owned networks. It enables network administrators of an organization to configure a virtual network topology, provision lightpaths, and allocate resources to end-users or other organizations. The software can manage optical networks with different underlying switching technologies. Network resources are abstracted as software objects in the system in order to achieve technology transparency. Currently, the software is used to manage CA* net 4, a Canadian Internet research and education optical network. Research institutes often lease resources from CA* net 4. Subsequently, administrators of an institute use the software to create end-to-end connections within their domain for the purpose of transferring large amount of scientific data for a period of time.

However, the UCLPv2 software does have some limitations. One is the lack of an automatic mechanism to establish an end-to-end connection across multiple

administrative domains using advertised resources of these domains. At present, representatives of organizations involved have to negotiate and coordinate the steps. This manual process will become very inefficient as the need for setting up inter-domain connections increases. To address the inter-domain routing problem within the context of UCLPv2, this thesis focuses on a solution based on the hierarchical inter-domain management approach proposed by Professor Bochmann [1]. The implementation framework is built upon Web services technology in accordance with the service-oriented approach of UCLPv2. The inter-domain routing is achieved by the collaborations of routing Web services on behalf of condominium networks owned by different organizations. Furthermore, a client program was implemented to provide end-users a graphical interface to request end-to-end connections across different domains.

The thesis is organized as follows. Chapter 1 gives an introduction of the paper. Chapter 2 provides background information on service-oriented architecture and Web services technology. Chapter 3 reviews some common Internet routing protocols and explains the Dijkstra's algorithm in detail. Chapter 4 presents the three-layer system architecture and the intra-domain routing capability of the UCLPv2 software. In addition, the chapter briefly explains the Eclipse Rich Client Platform upon which the UCLPv2 GUI was built. Chapter 5 describes the concepts of hierarchical inter-domain routing for networks with condo switches in detail. Furthermore, it explains how to apply the concepts to the design of the inter-domain routing solution within the context of UCLPv2. Chapter 6 focuses on the implementation of the hierarchical inter-domain routing. Finally, chapter 7 concludes the thesis and presents some ideas for future improvement.

Chapter 2 - Service-Oriented Architecture and Web Services Technology

2.1. Service-Oriented Architecture

Service-oriented architecture (SOA) evolves from the client-server computing model. The design principles of SOA dictate the software components to be modeled as services conforming to well-defined interfaces. An interface definition specifies what functions the component provides and how to interact with the service. The emphasis is on the design of interfaces, rather than component implementation. After deployment, services can be invoked over networks by any application knowing the interface [3]. In fact, a large software system can be built based on many independent services, which is the approach adopted by UCLPv2.

Service-oriented architecture is a natural extension of the object-oriented design. Object-oriented design can be viewed as an abstraction at the class level; however, SOA extends the abstraction higher to the service level. Consequently, SOA has the same benefits of the object-oriented design approach. One of the advantages of object-oriented design is the increased modularity. This is achieved by representing a real-world entity using a class definition which encapsulates the entity's attributes and behavior. SOA leverages on the object-oriented design approach by defining service interfaces first. The implementation logic is well encapsulated in the classes, and can be changed independently. Increased modularity implies less dependency between components, as a result, the whole system becomes loosely coupled.

One of the benefits of loosely coupled systems is their adaptability to changes in the future. Giving the fact that the nature of the software development process is iterative, and new feature requirements are inevitable, designing loosely coupled systems is of paramount importance.

The architecture promotes the reusability of software components. The same service can be invoked by different applications, while providing the same external behavior in accordance with its interface definition. Therefore, less programming codes are produced. The objective is to code once, and use it many times. SOA often consists of three participants: service requestor, service provider, and service registry [3]. Figure 1 shows their relationship.

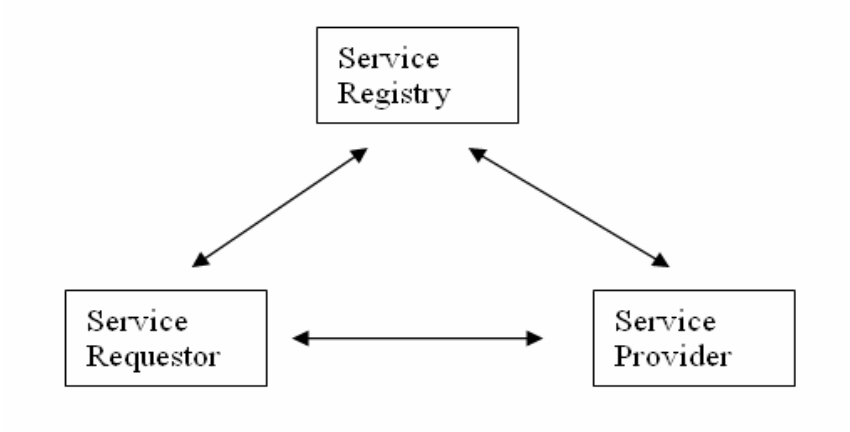

Figure 1: Service Participants Relationship [3]

Service requestor is the consumer of the service. It can communicate with the service registry to obtain a proxy of the service, and invokes the service through the proxy. The format of the request message is also defined according to the service interface description. The invocation can be synchronous or asynchronous depending on the application.

The service provider defines the interface, and publishes it to a well-known service registry. Furthermore, it is responsible for implementing the service. Upon invocation, it processes the input message and executes the implementation logic. It can optionally send back a response message to the requestor.

The service registry provides a common directory service and it acts as a broker between service provider and service requestor. A good example of a service registry is the Jini lookup registry. A service provider can register its service with the lookup registry by one

of two ways. If the service is implemented as a single class, the provider can store an instance of the service at the registry. If the implementation of the service involves multiple classes implemented on a server, then the provider can store a service proxy at the registry. The proxy communicates with other server-side objects. A client can send a request to the registry to search for a service with matching criteria such as service attributes. The registry responds with either a copy of the service instance or the service proxy.

Another well-known service registry is Universal Description, Discovery, and Integration (UDDI). UDDI acts as a centralized service registry storing the locations and descriptions of services. A client program can query the UDDI to obtain some meta-information about a desired service. However, the disadvantage is that developers have to program the service proxies themselves. At the current stage of development, UCLPv2 did not use a service registry, instead some Web service proxies were developed which the GUI client program uses to invoke the UCLP Web services. The proxies are to be installed along with the GUI program. The approach simplifies the whole system.

2.2. Web Services Technology

Web services technology is the most dominant implementation of SOA, and its standards are well accepted by the industry. The World Wide Web Consortium (W3C) working group defines a Web service as a software system designed to support interoperable machine-to-machine interaction over a network. Conceptually, services can be viewed as resources that provide an abstract set of functionalities. A Web service is not bound to any particular implementation. In practice, it has an interface described in a machine-processable format, using Web Service Description Language (WSDL). Other systems interact with it in a manner prescribed by its WSDL description using Simple Object Access Protocol (SOAP) messages, typically conveyed over HTTP with an XML serialization in conjunction with other Web-related standards [4].

The most important benefit of the Web services technology is that it enables the interoperability among software applications adhering to the relevant set of Web Services specifications. Software applications can be implemented on different platforms and using different programming languages. Figure 2 illustrates the Web services architecture stack. At the bottom of the stack is the communication layer. Transport protocols such as HTTP, FTP, JMS can be used to transfer Web service messages, which are encapsulated in SOAP envelops. The description layer uses WSDL standard to specify service interface definitions. The process layer includes several specifications to orchestrate Web services. The technologies underpinning Web services are XML, DTD, and XML Schema. The following sections discuss these technologies in more detail.

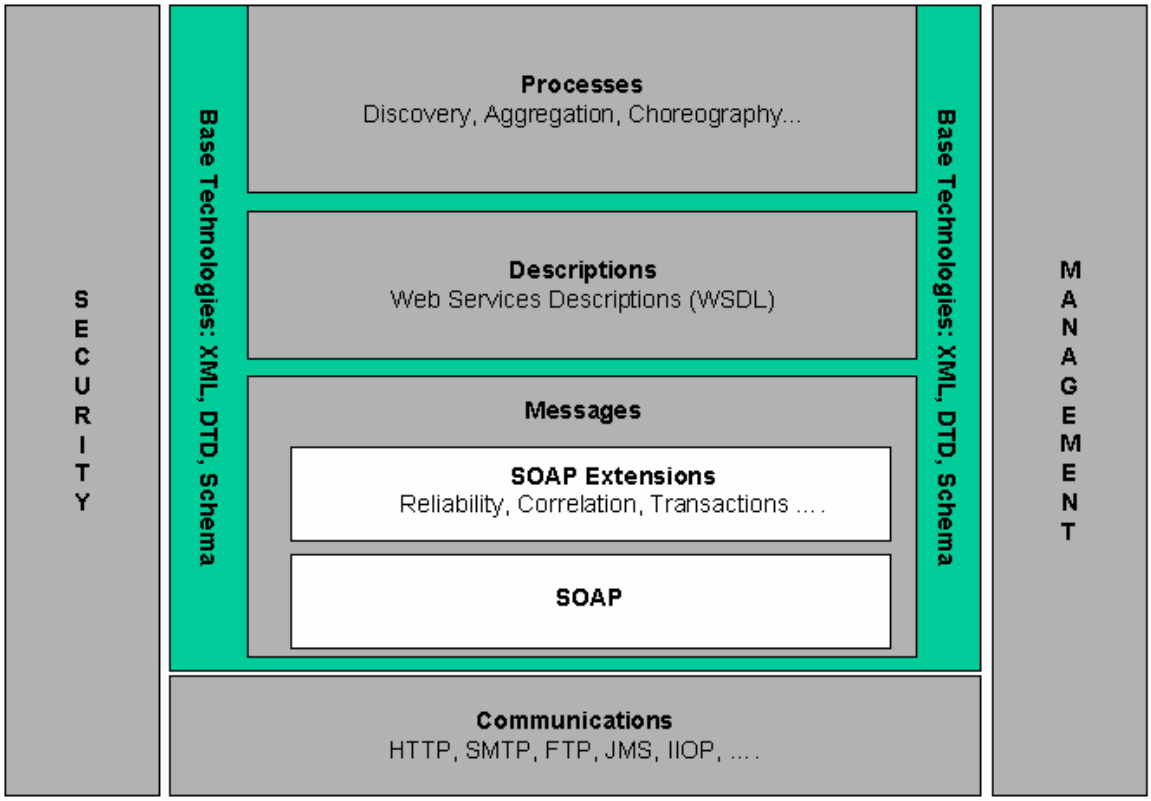

Figure 2: Web Services Architecture Stack [4]

2.2.1. Extensible Markup Language

The development of Web services technology started with XML, which stands for Extensible Markup Language. A markup language consists of a set of special tags that are intermingled with the primary data, and these metadata contain extra information describing the data.

XML has evolved into the de facto industry standard for structuring, describing, and exchanging textual information. XML adopts a tree structure to embed markup elements in document. Furthermore, it even allows users to design their own customized markup language for their documents.

XML-based documents achieve reusability by defining namespaces for elements. As a result, elements with the same name, but different namespaces, can be used in one document. The full-qualified name of any XML element consists of its namespace and its local name. Namespaces are based on Uniform Resource Identifier (URI) [3]. URIs are used to uniquely identify resources on the Web. Resource can be network accessible such as a web page, or network inaccessible such as an abstract concept. In fact, anyone can create a URI to represent anything as long as the general specification of URI is followed [5].

2.2.2. XML Schema

In order for an XML document to be machine-processable, it is not sufficient for the document to merely comply with the rules of the XML syntax. The document must also conform to a predefined structure, so that software can validate the content of the document and automatically process it. The XML Schema notation was designed to address the need for specifying the document structure.

XML Schema is a meta-language for describing the structure of XML documents and provides a data typing system to define vocabularies embedded in XML document. It has a built-in type hierarchy as shown in Figure 3. The base of the hierarchy is anyType, which is then extended into two groups: any simple type and all complex types.

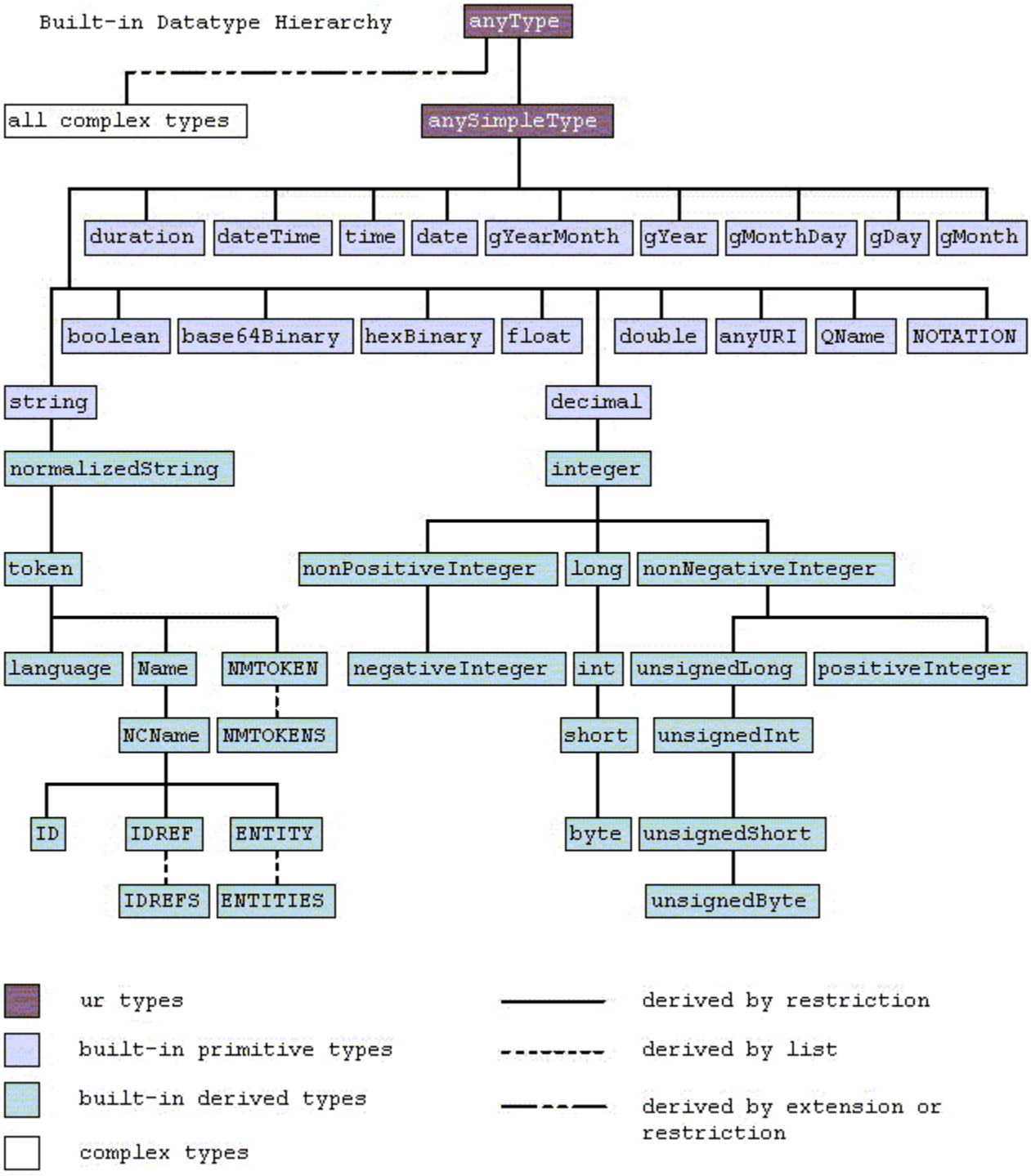

Figure 3: XML Schema Data Types [6]

Simple types including string, integer, and date which are built-in types. All complex types are user-defined. One main difference is that complex data types may contain nested XML elements and attributes, simple type can not have any child XML elements. Furthermore, one can specify sequence order and multiplicity of child elements in the definition of a complex data type.

Data types in an XML Schema are connected by derivation, that is, the definition of a new type must be based on another existing type. The two common techniques of derivation are restriction and extension. Restriction means narrowing the allowed set of values in the current type; whereas, extension means expand the set of values of an existing type the new type is based on [7].

The following is an example of defining a simple data type called countryType.

```
<xs:simpleType name="countryType">
        <xs:restriction base="xs:string">
                <xs:enumeration value="Canada"/>
                <xs:enumeration value="China"/>
                <xs:enumeration value="Columbia"/>
        </xs:restriction>
</xs:simpleType>

<element name="country" type="countryType">
```

We define the countryType by restricting the base type String. A valid value must be one of the strings Canada, China, Columbia. Consequently an XML document may contain the following XML element:

```
<country>Canada</country>
```

The following is an example of defining a complex data type that we used in our implementation.

```
<xs:complexType name=”routingEndPoints”>
        <xs:restriction base=“xs:anyType”>
                <xs:sequence>
                        <xs:element name="endPointOne" type="xs:string"
                        minOccurs="1" maxOccurs="1"/>
                        <xs:element name="endPointTwo" type="xs:string"
                        minOccurs="1" maxOccurs="1"/>
                </xs:sequence>
        </xs:restriction>
</xs:complexType>

<element name=“endPoints” type=“routingEndPoints”>
```

Consequently, an XML document may contain the following XML element:

```
< endPoints >
        <endPointOne>switch1</endPointOne>
        <endPointOne>switch5</endPointOne>
</endPoints >
```

2.2.3. Web Service Description Language

The W3C organization defined WSDL as a language for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information [8]. In essence, it provides the functional description of a Web service, and plays the same role for Web services as the Interface Definition Language (IDL) for Common Object Request Broker Architecture (CORBA) [9].

WSDL uses XML elements to describe the four aspects of a Web service which are the functionalities of a Web service, the data types used in the request and response

messages, the deployment location of a service, and the binding to a network transport protocol. In other words, it specifies what the service does, where to locate the service, and how to access the service. Since WSDL is a XML-based common language, it is platform-neutral and language-independent. Figure 4 shows a typical WSDL file structure.

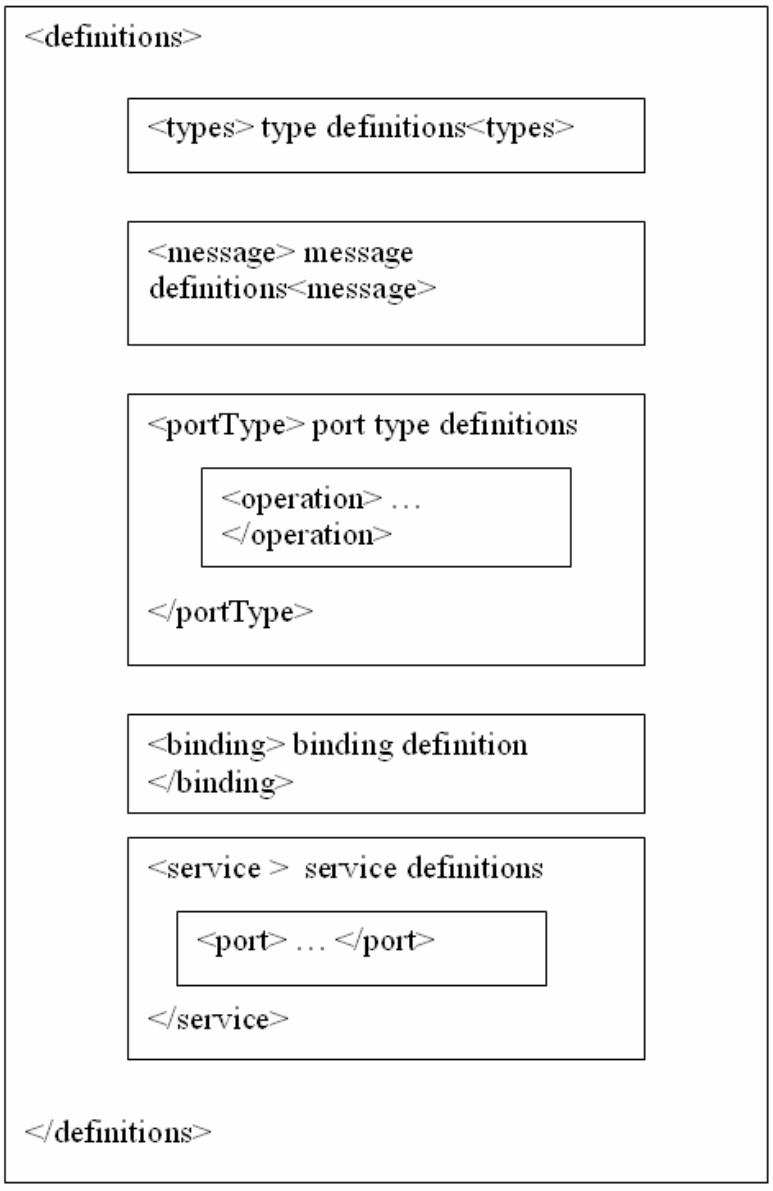

Figure 4: WSDL File Structure

A WSDL document uses the following keywords to describe the service [8].

- Definitions: It is the root XML element, signifying the starting point of a WSDL definition. It specifies the name of a Web service, and namespaces used in the WSDL file.
- Types: It is a container to define data type definitions used in the request and response messages. A commonly used type system is XML Schema; however, it is not confined to any particular type system, thereby providing the maximum flexibility.
- Message: This section defines the precise format of input and output messages. It consists of the name of the message, and zero or more message parts. The message part specifies the input and output parameters.
- PortType: This section comprises one or more operations. It describes all the actions that can be performed by a Web service. In other words, it specifies the operational interface of a Web service.
- Operation: Each operation is equivalent to a message signature. An operation can specify input and output messages by referencing their unique names.
- Binding: It specifies a concrete protocol and data format to be used for a particular port type. As a result, abstract input and output messages can be mapped to concrete data representation on the wire. A common binding is to use SOAP message format over HTTP protocol.
- Service: It consists of one or more ports.
- Port: A port is an endpoint that specifies the network address of a service and the corresponding binding.

There are two additional utility elements.

- Documentation: its purpose is to provide some annotation for human readers.
- Import: the purpose of this element is to advocate reusability of WSDL documents. It allows one WSDL document to import other WSDL documents and use any elements already defined in those namespaces.

Message exchange is an importance aspect of Web services. Messages represent the data structure and can be enclosed in different protocols. Common examples are HTTP GET/POST message, a plain XML-message, SOAP message and so on. Among them, SOAP messaging has become the most prevalent means for communication with Web services.

2.2.4. SOAP Protocol

Simple Object Access Protocol is a protocol developed by W3C. It is an XML-based communication protocol that enables distributed applications to exchange structured and typed information. It is platform neutral and language independent. Due to its simplicity and extensibility, it has become a cornerstone of the Web service architecture. Figure 5 shows the structure of a SOAP message.

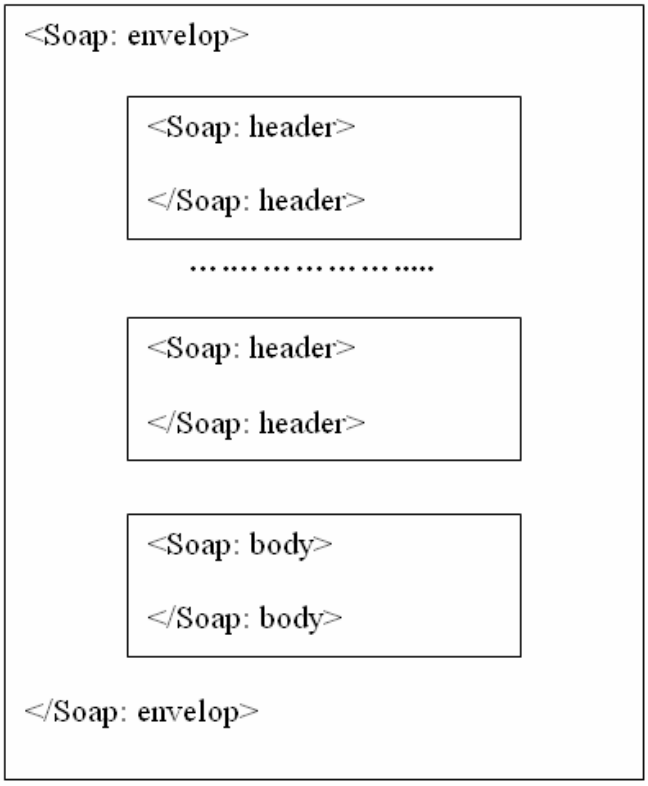

Figure 5: SOAP Message Structure

A Soap message uses keyword “envelop” to encapsulate a header section and a body section. The header section can contain zero or more headers, whereas, there can only be one Soap body per message. The body consists of the data content to be exchanged between applications. The purpose of the header is to provide extensibility to the Soap mechanism. For example, one can include some security information in a header without affecting the format of the message in the body section. Furthermore, header attributes serve to inform the message recipients to process the information in a particular way. Depending on the content of the header, new functionality can be added with minimum impact on the current system [10].

The SOAP data model and SOAP encoding provide means to map application-level data structure to XML representation on the wires. For example, Java objects can be transformed into XML elements and sent down the wire, on the receiving end, the XML data will be converted back to the Java representation.

The SOAP specification also includes a protocol binding framework. A binding implies a set of rules for transferring a SOAP message on top of a transport protocol. One of the most popular binding is SOAP over HTTP, which takes advantage of the request-response mechanism of HTTP to achieve two-way communication. In summary, SOAP describes the format of the envelope, but not the format of the XML data inside the envelope [11].

2.2.5. Axis Web Service Engine

Apache Axis is a popular SOAP engine, which is an implementation of the SOAP protocol. It is an open-source software implemented both in Java and C++ languages. It can be used to host Web services, process SOAP messages, and manipulate WSDL documents. Furthermore, it provides client-side APIs to invoke Web services, and utilities to deploy Web services. Its set of tools enables the fast development of Web services both on the client-side and server-side.

Figure 6 shows a simplified view of the Axis architecture, which is based on the concept of chains of message handlers. A message handler in Axis is a software component that implements a predefined interface. When a SOAP message is received, the engine will pass the message through a chain of handlers, with each carrying out specific functions. One advantage of the architecture is that developers can freely add their own handlers to the processing chain. UCLPv2 took advantage of this feature by adding a customized security handler into the chain to check the authenticity of messages. The core business logic pertaining to a Web service is typically implemented in one of the handlers [3].

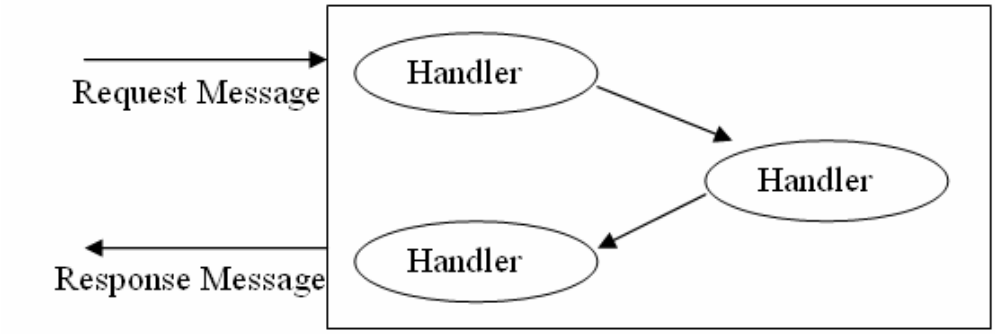

Figure 6: A simplified View of Axis Architecture [3]

2.2.6. Service Orchestration and Business Process Execution Language

To achieve the full potential of the Web services technology, service orchestration plays an important role. Service orchestration focuses on the composition and interaction of Web services in order to realize a common business objective. The Business Process Execution Language for Web Services (BPEL) is developed for this purpose. BPEL uses a XML-based grammar to describe the behavior aspect of a business process. It enables developers to define an executable long-running BPEL process which interacts with third-parties' Web services in a deterministic order. The service invocations can be coordinated in sequential or concurrent order. Furthermore, a BPEL process itself is a Web service, and exposes its interface defined by a WSDL document. As a result, service orchestration becomes recursive in nature.

A BPEL process is implemented as an XML file that consists of BPEL specific constructs in order to define the process flow. These XML constructs include partner definitions, message receive, message reply, variable manipulation, flow control, sequential invocation, concurrent invocation, exception handling, and transaction compensation. For example, the following are some commonly used elements: <receive>, <reply>, <invoke>, <sequence>.

A BPEL specification uses WSDL message definition and XML Schema data types to model information. For example, request and response messages are defined in WSDL documents, and referenced in BPEL process, so long as the WSDL files are imported into the BPLE process.

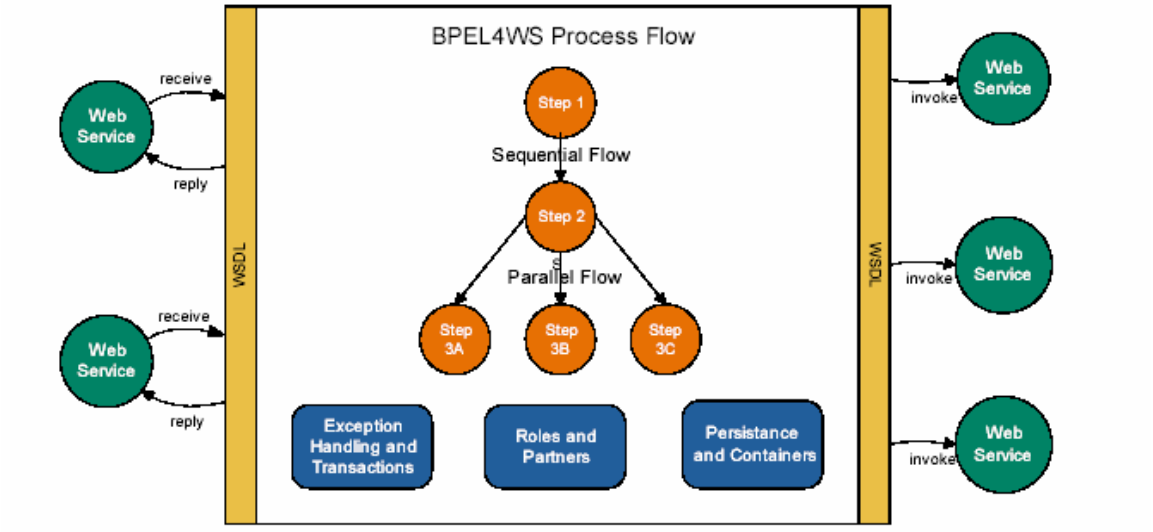

Figure 7: BPEL Process Example [12]

Figure 7 illustrates a typical scenario. A running BPEL process is invoked by a Web service. Upon receiving a request message, the BPEL process creates an instance of itself, which executes the control logic defined in the XML file. The steps consist of invoking a series of external Web services. The activities can be carried out in both sequential and concurrent orders. At the end of the lifecycle of the instance, a reply message is

constructed and sent back to the client. The message could indicate a successful result, or an exception. Meanwhile, the process waits for the next invocation.

The BPEL XML files are deployed in a BPEL engine, which interprets and executes the BPEL grammar. The UCLP project adopted an open source BPEL engine, namely ActiveBPEL. It provides a robust runtime environment that is capable of executing process definitions defined by the BPEL specifications [13].

Chapter 3 - Overview of Internet Routing Protocols

Routing protocols are developed to facilitate the process of computing paths from a source node to a destination node in a network. In fact, routing protocols exist in telephone network predating the Internet. As the Internet grew into thousands of inter-connected networks of different sizes, a number of routing protocols were designed to meet different requirements.

A routing protocol constructs a routing table at each router. This is achieved by the exchange of routing information, which reflects current traffic conditions in the network. When a packet arrives at a router, a switching process takes place. The router reads the destination of the packet, and looks up the corresponding entry in its routing table. It then forwards the packet to the next hop found in the table.

Routing protocols can be categorized into intra-domain and inter-domain based on autonomous system division. Intra-domain protocols perform routing functions within a single administratively autonomous domain. On the other hand, inter-domain protocols carry out routing functions between different autonomous domains.

Another categorization of routing protocols is based on the type of the distributed routing algorithm. These include distance-vector, link-state and path-vector routing. A router using the distance-vector protocol sends its neighbors a vector of its estimated distances to all the subnetworks periodically. The router builds up its routing table based on the received updates from its neighbors. However, it does not have the knowledge of the topology of the network. Routing Information Protocol (RIP) is an example of the distance-vector approach. In a link-state protocol, every router sends the cost values of its own links to all other routers in the network. As a result, every router knows the topology of the network. By running a shortest-path algorithm, a router can build up its routing table. Open Shortest-Path First (OSPF) protocol is an example of the link-state approach. The path-vector approach is quite different, it does not use routing metrics, instead it is based on routing policies associated with autonomous systems (AS). The protocol

achieves better scalability and is often used for inter-domain routing. Border Gateway Protocol (BGP) is an example of the path-vector approach.

This chapter begins with a brief overview of RIP as an example of intra-domain protocol and BGP as an example of inter-domain protocol. It then describes OSPF including Dijkstar's algorithm in detail in order to relate our hierarchical inter-domain routing approach with some fundamental concepts in OSPF.

3.1. Routing Information Protocol

RIP is a commonly used protocol based on a distance vector algorithm. The algorithm uses distance vectors to mathematically compute lowest-metric paths to all given destinations. The distance metric in RIP represents the number of hops between two network elements. One main characteristic of the protocol is that routing information is exchanged only between adjacent routers. Furthermore, the protocol is suitable for medium-size networks. This is due to the mechanism adopted to prevent routing loop. Any destination with a hop count greater than 15 is considered unreachable.

The way RIP works is quite straightforward. Each router maintains a routing table that has a single entry for every destination subnetwork. Each entry consists of information such as distance to the identified network and the IP address of the next hop router. Each router sends an update message describing its routing table to all its neighboring routers at regular intervals or when the network topology changes. If a receiving router discovers that a shorter path to a destination exists, it will update its routing table. By repeating the update process at every router, any network change is quickly propagated to the entire network. In contrast to the OSPF protocol, routers using RIP do not have a complete view of the network topology [14].

Over the years, RIP has evolved to version 2 with enhanced capabilities such as authentication, subnet masks, and multicast support. Since RIP assumes that the same

routing metric is used by all routers, it is only suitable for routing within a single Autonomous System. In addition, the protocol is still not suitable for large network due to the 15 hop limitation. Furthermore, "This protocol uses fixed metrics to compare alternative routes. It is not appropriate for situations where routes need to be chosen based on real-time parameters such a measured delay, reliability, or load. The obvious extensions to allow metrics of this type are likely to introduce instabilities of a sort that the protocol is not designed to handle" [15].

3.2. Border Gateway Protocol

BGP is an exterior routing protocol which has become the most widely used protocol for inter-Autonomous Systems routing. It can also be used for the purpose of intra-AS routing. Depending on the mode of operation, it can be referred to as Internal BGP or External BGP. BGP-4 is the newest version. One interesting fact is that BGP uses TCP as its transport protocol whereas most other protocols use UDP.

BGP thrives on a quite different routing approach, that is, instead of using routing metrics, it is based on lists of autonomous systems to be traversed to reach a destination network. This coarse-grained approach improves scalability, thereby, making it suitable for large network such as the Internet. The selection of a path can be subject to routing policy. For example, one common policy is to choose a path that minimizes the number of ASs traversed. Another policy could be to avoid particular AS due to performance reasons [14].

Similar to other routing protocols, BGP relies on the exchange of routing information among routers. However, the roles of these routers are two-folded. First, the router must run an intra-domain routing protocol to communicate with other routers within the same AS in order to acquire the topology of the AS. Secondly, it runs BGP to exchange information with routers located in different ASs.

BGP routers must establish neighboring relationship with routers in other domains first. A router initiates the neighbor acquisition process by sending an Open message to another router, which can respond with a Keepalive message. Subsequently, neighboring routers must send Keepalive messages to each other periodically in order to maintain the relationship. Routers will broadcast Update messages when their routing table has changed. An Update message consists of new route and/or withdrawn route information. A route entry includes a list of ASs that are traversed for this route, the IP address of the next-hop router for this route, and a list of subnetworks that can be reached. When another router receives the Update message, it will compare a new route with existing information in its routing table, and decide if the new route is the preferred path to the subnetworks. If so, it will add its current AS to the AS list, modify the next-hop router, and forward the message to its neighbors. In addition, it will update its routing table accordingly. To prevent routing loops, a router ignores an Update message that already includes its own AS in the AS list [14].

3.3. Open Shortest-Path First Protocol

OSPF is a link-state protocol and is specifically designed for intra-AS routing. Since each router is aware of the state of its physical links, it will send the state information to all the routers within its area. On subsequent link state changes, each router will again flood the network with update messages. This approach results in every router to maintain a complete view of the network topology in its routing database. Based on the consistent view of the network topology, each router runs the Dijkstra's algorithm to calculate shortest paths between every pair of source and destination [14].

OSPF achieves better scalability by introducing the concept of subdividing a large network into smaller, more manageable subnetworks or areas. The idea is to connect all areas to a single area known as the backbone area. Acting as the core of a network, the backbone area performs inter-area routing through the use of border area routers. The role of a border area router is to connect the backbone area with one or more other areas;

therefore, a border area router can be owned by multiple areas [14]. The role of a condo switch in a UCLP network is very similar to a border area router in OSPF, since a condo switch is also used to connect different networks [1]. Routers in a non-backbone area uses OSPF as their intra-area routing protocol. By limiting the exchange of routing information to only routers within the same area, the overall routing traffic is reduced.

We can also view the overall network in OSPF as having a two-level hierarchical structure, with the backbone area at the top level. Our hierarchical approach for inter-domain routing for network with condo switches shares some similarities with the OSPF hierarchical concepts as explained in Chapter 5.

3.3.1. Dijkstra's Algorithm

Dijkstra's algorithm uses a single metric referred to as link cost to calculate the shortest paths from a given vertex to all other vertices in a directed graph [16]. The link cost must be nonnegative. The algorithm processes every vertex in a graph starting from the source vertex. It divides vertices in a graph into two sets. Set A includes all vertices to which we have already found the shortest paths from the source vertex; in addition, the total cost of the path is associated with each of these nodes. Set B includes the rest of the unresolved vertices.

The following description of the algorithm uses the following symbols. Symbol s represents the source vertex. Symbol u and v represent any vertex in the graph. C(v) represents the cost of the shortest path found so far from s to v. In case that the path between s and v does not exist, C(v) would be set to infinity. Symbol L(u,v) represents the link cost from vertex u to v. We use a minimum priority queue to implement B with the order of the vertices in the queue determined by C(v).

The algorithm initializes C[s] to zero, and the path costs of all other vertices to infinity. The set A is empty, and set B contains all the vertices including the source. The algorithm repeatedly chooses a vertex u with the minimum cost value from B, and moves the vertex

to set A. It then applies an edge relaxation operation on every adjacent vertex of u as follows. Suppose v is one of the adjacent vertices of u. If the sum of C(u) and L(u,v) is less than C(v), that means a shorter path from s to v has been found, then the current value of C[v] should be replaced with the sum [17]. The pseudo code for Dijkstra's algorithm is shown below [18].

```
C[s] = 0;
C[u] = ∞ for all u ≠ s;
A ← Ø;
B ← all vertices;
while B ≠ Ø do
        u ← Extract-Minimum(B);
        A ← A union {u};
        for each adjacent vertex v of u such that v is in B do
                (perform edge relaxation operation)
                if C[u] + L(u,v) < C[v] then
                        C[v] ← C[u] + L(u,v);
```

We will illustrate how the algorithm works with an example shown in Figure 8.

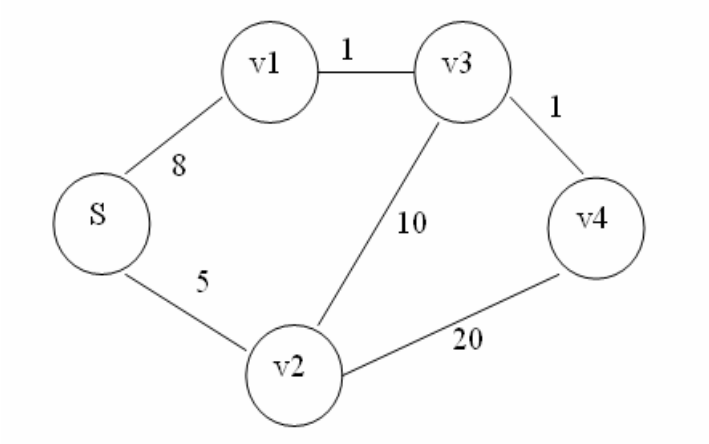

Figure 8: Graph Example

Step 1: The algorithm starts with initializing the following values.

C(s) := 0; C(v1) := C(v2) := C(v3) := C(v4) := infinite;

A := {}; B := {s, v1,v2,v3,v4}

Step 2: It extracts a vertex with the minimum cost from B. Since the source vertex s has a cost of zero, it will be chosen and added to A. The edge relaxation operation will be applied on the adjacent vertices of s, which leads to

C(v1) = 8; C(v2) = 5; C(v3) := C(v4) := infinite;

A := {s}

B := {v1, v2, v3,v4}

Step 3: It repeats the step 2. The vertex with minimum cost extracted from B is v2. Its adjacent vertices are s, v3 and v4. Since s is already in A, the edge relaxation operation will be applied on v3 and v4, which leads to

Given: C(v3) = infinite; C(v2) = 5; L(v2,v3) = 10;

Therefore: C(v3) > C(v2)+L(v2,v3)

C(v3) := 15;

Similarly C(v4) := C(v2)+L(v2,v4) := 5+20 := 25;

C(v1) = 8; C(v2) = 5;

A := {s, v2}

B := {v1, v3,v4}

Step 4: The algorithm repeats the step 2 until B is empty.

C(s) := 0; C(v1) := 8; C(v2) := 5;

C(v3) := 9; C(v4) := 10;

A := {s, v1, v2, v3, v4}

B := {}

The algorithm finds the cost values of the shortest paths from the source to all other vertices in the graph. In practice, it is often necessary to record the corresponding shortest paths as well. This can be achieved by using a hash table to store a vertex and its preceding vertex as a key and value pair. Whenever the edge relaxation operation is

successfully applied to a vertex, the hash table will be updated with its new preceding vertex.

An infinite cost value indicates that a path does not exist. If we are searching for a shortest path between only two vertices, then the algorithm can exit when the destination vertex is processed.

Chapter 4 - User Controlled Lightpath Provisioning System Version 2

4.1. UCLPv2 Overview

There is an increasing demand for organizations to acquire network resources from multiple service providers or domains and dynamically control these resources suiting the data transfer requirement of their applications. These applications are often data intensive, and require the dynamic establishment of multiple end-to-end connections for a period of time. The end-users of the organizations desire the flexibility to re-configure the end-to-end connections, and full control and management of their resources without the interference of network operators. Consequently, a user controlled lightpath provisioning system was envisioned and implemented. UCLPv2 provides a software solution to enable end-users to provision network resources in the form of lightpaths and dynamically configure network topology subject to the requirements of user applications. Furthermore, users can control bandwidth usage by partitioning a single lightpath into multiple lighpaths with smaller capacity, or vice versa. Hence, the software provides a more efficient means for resource utilization. It also facilitates an organization to sublease resources to other organizations. In essence, UCLPv2, based on SOA design approach, can be viewed as a configuration and provisioning tool for optical networks. However, it is not intended to be an automated management system [2].

A key concept in UCLPv2 is the definition of a fundamental lightpath. A fundamental lightpath is defined as an abstract representation of the basic unit of optical network partitions. It implies a dedicated communication channel between two adjacent switches. As a result, it can represent a SONET channel, a wavelength in Wavelength Division Multiplexing (WDM) network or an Ethernet channel and so on, making this approach technology independent. It is especially beneficial when making an end-to-end connection across heterogeneous networks. Furthermore, a lightpath always has two end-points. These end-points can be physical ports or virtual ports of optical switches [19]. By

modeling physical resources as software objects, the system enables users to provision lightpaths based on physical links.

Another salient feature is that the system is service-oriented in the sense that resources including switches, lightpaths, and interfaces are exposed as and managed through Web services. As a result, users can easily manipulate these resources through the use of Web service orchestration tools, such as BPEL. More importantly, by linking application Web services with network resource Web services in BPEL workflows, users effectively bring the network into the application itself [2].

The software introduces a new concept called Articulated Private Network (APN). An APN comprises a collection of network resources and is similar to a virtual private network, except its network topology can be changed or articulated dynamically by end-users [20]. The words "network topology" refers to the logical configuration of one or more end-to-end connections using resources within an APN. By executing an APN, end-to-end connections will be physically established.

There are three types of users in a UCLPv2 system: physical network administrator, organization administrator, and end-user. Different users have different level of privileges. A physical network administrator belongs to an organization that owns physical networks. A user with this role is granted all the operational rights and is responsible for creating and allocating logical resources to other organizations. An APN administrator belongs to an organization that does not own physical network but acquires some logical resources from a physical network provider. He or she is responsible for managing the resources. A typical operation is to create multiple APNs and assign them to different end-users. The software meets the needs of end-users of an organization by allowing them to execute APNs and to re-configure APNs. However, end-users cannot add or delete logical resources of an APN [20].

4.2. UCLPv2 Use Case

The following is a typical use case of the software. A physical network administrator uses the UCLPv2 GUI to create a model of the physical network as shown in Figure 9. The model consists of graphical icons representing network elements for example, a variety of optical switches. The administrator then provisions each network element by allocating bandwidth for UCLP usage. Furthermore, these elements are connected by lines that correspond to the physical fiber links between switches. The topology of the model reflects the topology of the physical network. Subsequently, the administrator creates logical resources including lightpaths and interfaces. An interface represents a port or virtual port on a network element. A subset of the lightpaths and interfaces can be grouped into a resource list and given to an organization. The resource list is an XML file describing these logical resources, and it can be sent to APN administrators of the organization by any electronic means.

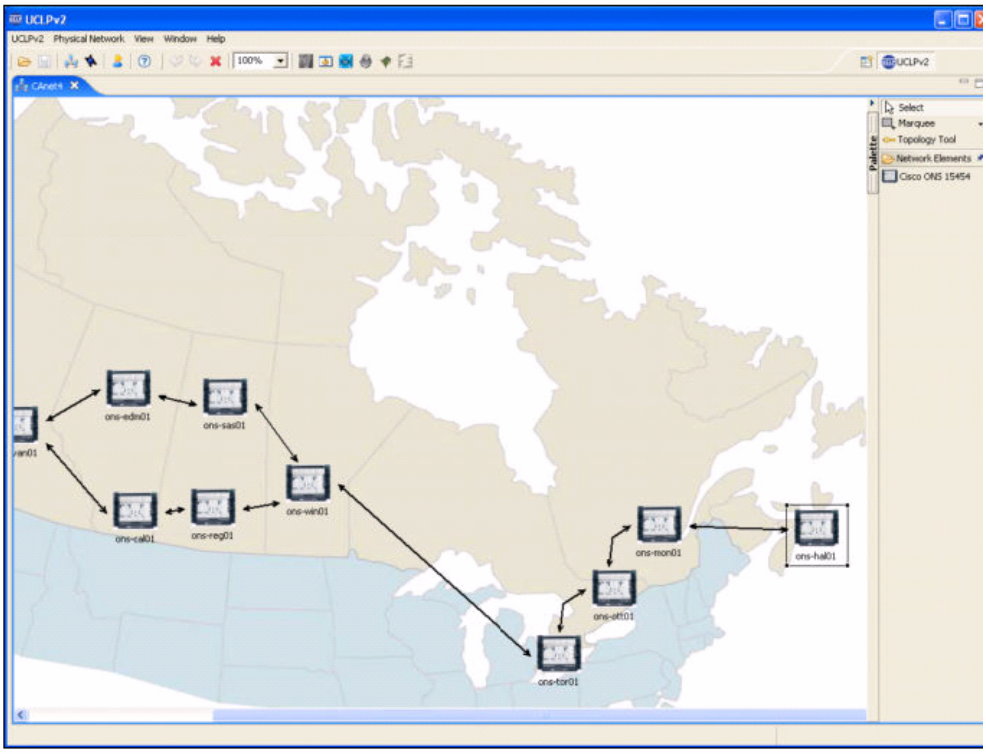

Figure 9: Physical Network View [21]

An APN administrator of the organization launches the GUI program that loads the resource list. A visual representation of the logical resources can be displayed in the resource list view of the GUI as shown in Figure 10. The administrator can group some resources into an APN with a pre-configured topology, and assign the APN to an end-user.

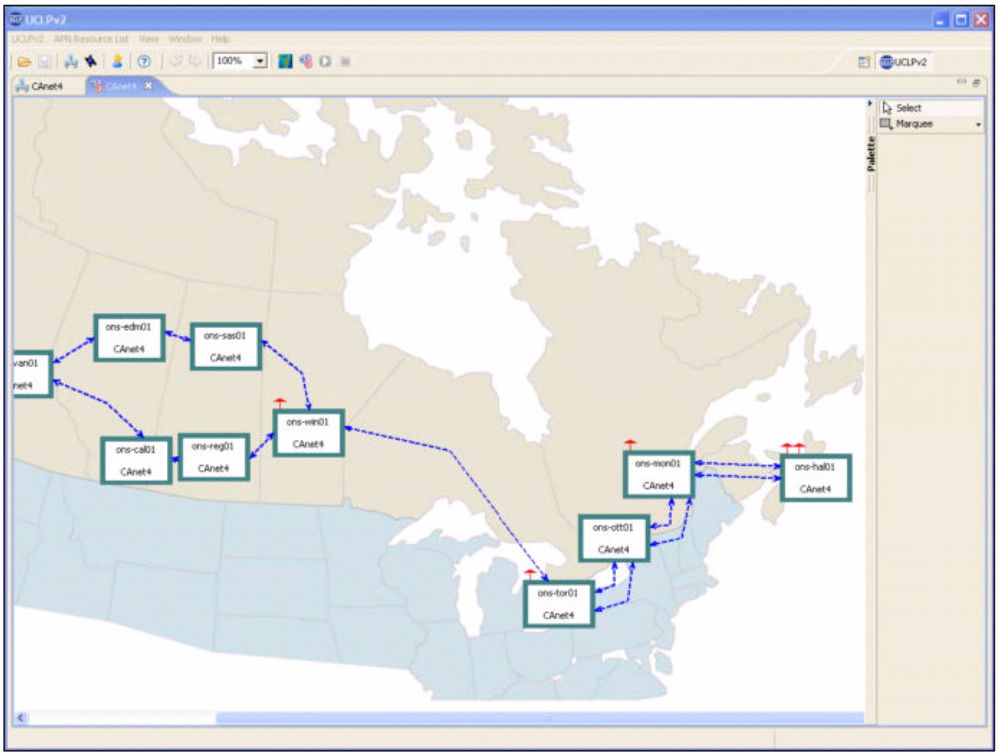

Figure 10: Logical Resource View [21]

The end-user uses the GUI to view, configure or execute an APN. Execution of an APN means the making of cross connections on the switches along end-to-end paths. Figure 11 shows an APN with two established end-to-end connections in different colors. After activating the topology, data transfer between any two end points can take place. In case of new applications that require different end-to-end connections, the user can un-execute the APN, change its topology, and re-execute it. In other words, the user can articulate the private network in a way suitable for the current work context.

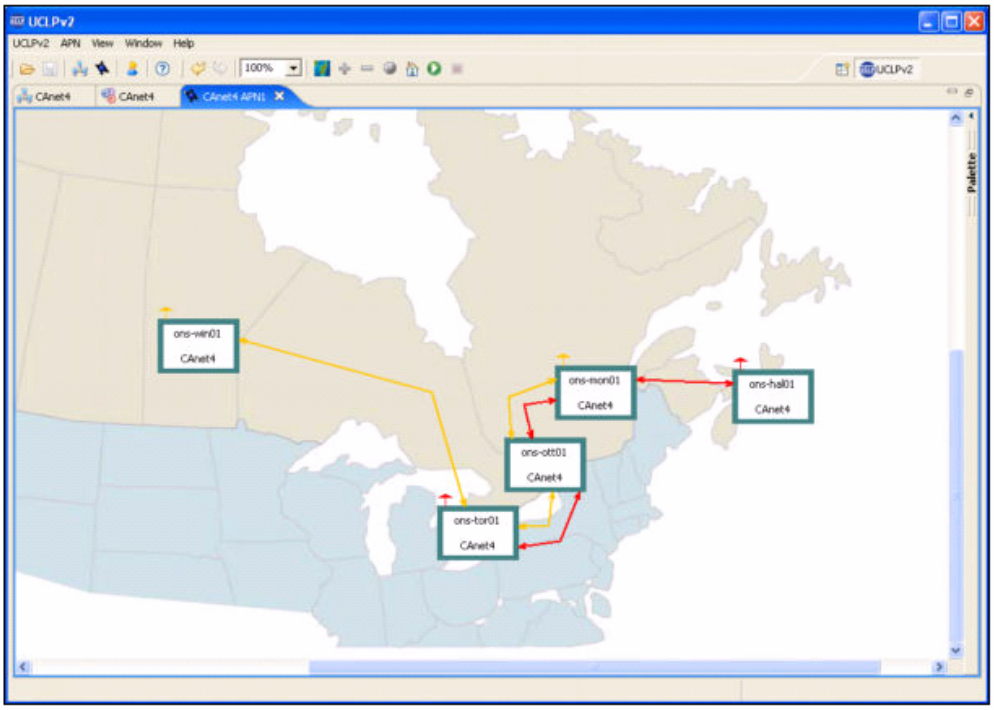

Figure 11: APN View [21]

4.3. UCLPv2 Architecture

The design and implementation of UCLPv2 leverages multiple technologies including Eclipse Rich Client Platform (RCP), Web services, and BPEL technologies. Java is the programming language chosen.

The architecture of the UCLP system is service-oriented and consists of three layers: User Access Layer, Service Orchestration Layer, and Resource Management Layer. Figure 12 shows the high-level architecture of the system.

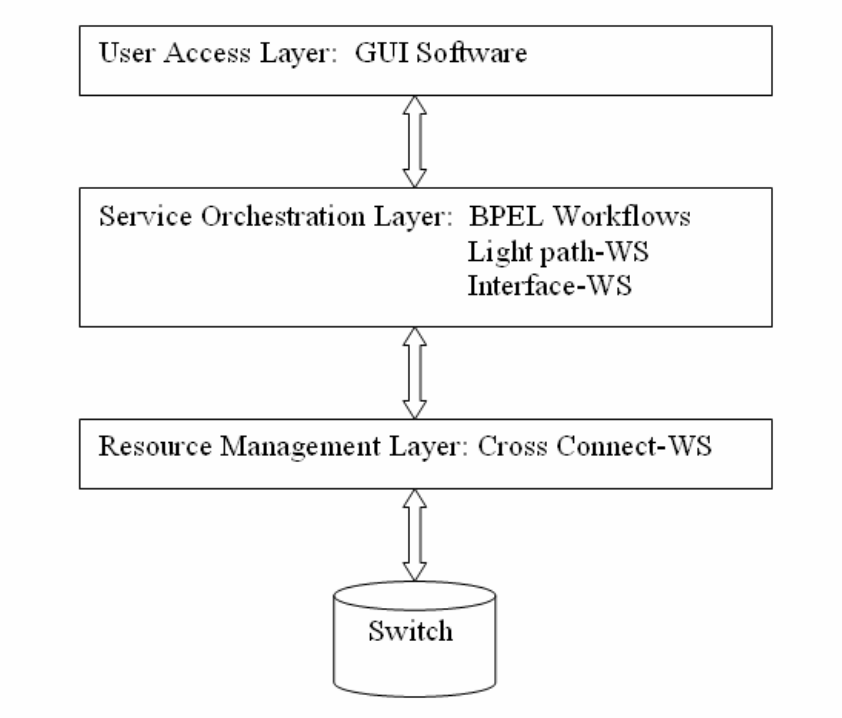

Figure 12: Three Layered System Architecture [20]

4.3.1. Resource Management Layer

The Resource Management Layer comprises Network Element Web services (NE-WS) that control a family of physical network elements such as optical switches. These are Axis-based Web services to be deployed on the network carrier's side. The main Web service implemented today is a Cross Connect Web Service (XC-WS). XC-WS is designed to manage optical switching devices such as SONET, SDH, Fiber, and Lambda Cross Connects. The service has a well-defined WSDL interface that describes different operations performed on switches. These operations include cross connecting and uncross connecting a switch. It also handles resource allocation by assigning channels to users and provides current state information about these resources. The state information indicates whether a channel is currently being assigned or used. Resource-related information is kept in a local database. The service translates XML requests into TL1/CLI commands in order to control physical devices. In essence, XC-WS is the lowest level of service interacting directly with physical devices. The Web services in the resource management layer can be extended in the future to include new services to

manage a GMPLS cloud, a VLAN enabled Ethernet switch, a layer-three router and so on [20].

4.3.2. Service Orchestration Layer

The objective of service orchestration layer is to provide an abstracted view of the lower-layer network resources. The idea is to use a high-level Web service to encapsulate the orchestration of a number of Web services implemented in the resource management layer; as a result, the details of invoking lower layer services become transparent to end-users or other applications. Since orchestration provides a flexible way to configure network topologies by invoking different Web services in different orders, it makes the provisioning of network resources more effortless [20].

The implementation of this layer harnesses the power of BEPL technology to perform Web service orchestration. With the use of BPEL workflow, it enables users to integrate multiple Web service invocations into the execution of a single process. Two basic Web services form the foundation of other higher-level workflows. These are Lightpath Web Service (LP-WS) and Interface Web Service (I-WS) [20].

A LP-WS represents a communication link or lightpath connecting two Web service-enabled end-points. An end-point can be an optical switch, an instrument, or any service-enabled device. Consequently, LP-WS hides the underlying heterogeneous network devices, thus achieving interoperability. Its WSDL interface definition specifies a set of operations performed on lightpaths. The main operations include lightpath concatenation, unlink, partition, and bonding. The concatenation operation creates a longer lightpath by connecting two existing lightpaths. It invokes the XC-WS to make the cross connection on the common switch of the two lightpaths. The unlink operation has the opposite effect, it can be used to undo a concatenation. The partition operation divides the bandwidth of an existing lightpath into smaller channels, and creates child lightpaths representing these channels. The bonding operation is the opposite of partition. In addition, a LP-WS

provides operations to query the state of the lightpath. The state information indicates whether the lightpath is currently concatenated or bonded with other lightpaths [20].

By partitioning an optical network into fundamental lightpaths, and making these lightpath Web service enabled, higher-layer software can better manipulate network resources meanwhile hiding the complexity from end-users.

Figure 13 shows the relationship between LP-WS and XC-WS.

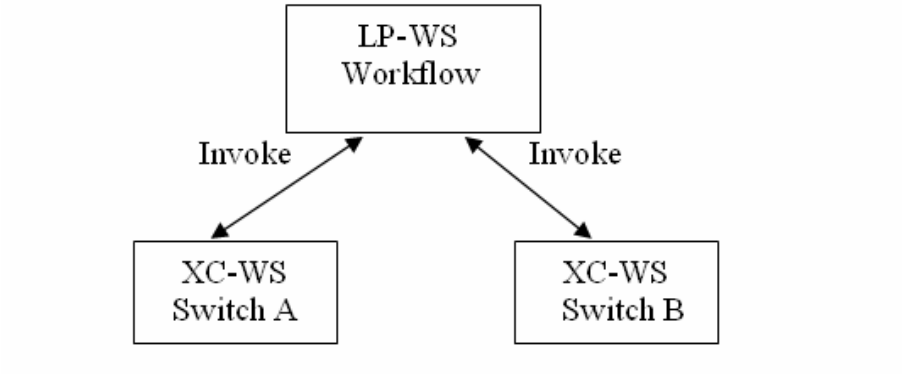

Figure 13: Relationship between LP-WS and XC-WS

A I-WS represents a single resource on a network element. For example, it can represent a physical port or a virtual port on a switch. It supports operations such as create, delete, query, and use. The purpose of this service is to provide interfaces on the switch for instruments to attach to. These operations invoke the XC-WS of the switch associated with a port [20]. Figure 14 shows the relationship between I-WS and XC-WS.

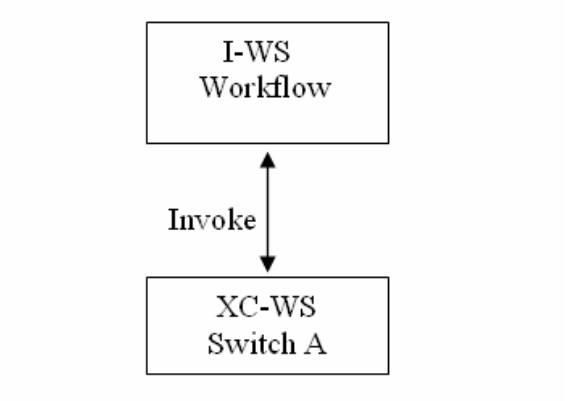

Figure 14: Relation between I-WS and XC-WS

Figure 15 shows an end-to-end path abstractly represented by constituent Web services, where LP-WSs represent logical channels between switches, and I-WSs represent logical ports on switch A and C.

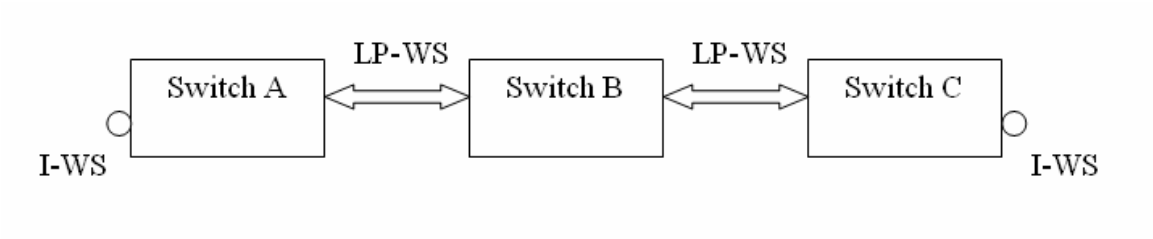

Figure 15: LP-WS and I-WS forming an END-TO-END Path

A user can establish an end-to-end connection from switch A to C by invoking I-WS and LP-WS services. To better automate this process, the service orchestration layer includes two higher-level workflows: utility workflow and custom workflow. The utility workflow, also known as connection management Web service, enables users to dynamically set up an end-to-end connection when provided with a list of lightpath and interface identifiers. Furthermore, a user can call the utility workflow to partition a single lightpath into many lighptaths with smaller granularity of bandwidth. The reverse function, bonding multiple smaller lightpaths into a single lightpath is also feasible [20].

The customer workflow enables a user to predefine any particular sequence for invoking web services. The result is a programmatically generated BPEL file, which represents a specific network topology of the APN. At any point of time, the user can execute the

workflow, that is, invoke the underlying LP-WS and I-WS to establish end-to-end connections conforming to the predefined topology [20].

4.3.3. User Access Layer

User Access Layer includes a graphical user interface (GUI) program that facilitates end-users to interact with the Web services in the orchestration layer. The GUI is built upon the Eclipse RCP which was chosen due to its extensibility and popularity.

Eclipse is an open source project initially developed by IBM. It can be used as an Integrated Development Environment or as a product base for developing new tools. The plug-in based architecture empowers developers to integrate new tools seamlessly with the existing Eclipse Platform and other tools. In Eclipse, a plug-in is a software component that provides a specific set of functions. The Eclipse model offers mechanism to discover, integrate, and run a set of plug-ins. There are two types of relationship between plug-ins, dependency and extension. A plug-in can specify what other plug-ins it requires in order to run. Furthermore, a plug-in can extend the capabilities of an existing plug-in. Eclipse RCP is a set of plug-ins that acts as a middleware for building rich client applications [22].

The Eclipse platform is built upon the SWT and JFace frameworks. SWT is comprised of a collection of primitive GUI components and is implemented entirely in Java, thus, it is portable cross different platforms. In addition, its design philosophy is to use native widgets as much as possible by invoking the operative system functions through Java Native Interface. As a result, the graphical performance is comparable to a native application, while keeping the same look and feel [23]. JFace is a toolkit developed by Eclipse team, and it is layered on top of SWT. Its purpose is to provide higher-level reusable components to solve common user interface problems. It frees developers from the tedious, low-level user interface programming. It adopts the Model-View-Controller (MVC) design pattern to transform SWT widgets into model-driven widgets. In addition, it extends the functionality of SWT by adding components such as wizards, dialogs and so forth [24].

By extending the Eclipse platform, UCLPv2 developers built an integrated tool consisting of a set of views and graphical editors. The views are used to display and organize network resources, which are categorized into physical, logical, and APN resources. The GUI provides both outline and detailed view of the resources. In addition, the RCP framework adds the navigation capabilities to these views [21].

The graphic editors include the physical network editor, the resource list editor, the APN editor, and the physical resource utilization editor. The physical network editor enables physical network administrators to create physical network representation, provision network elements and logical resources such as lightpath and interface Web services. The resource list editor enables physical network administrators and APN administrators to view, edit and manage their logical resources. Users can select some resources to create new resource lists, APNs or end-to-end connections. The APN editor enables all type of users to view the resources in an APN. Users can change the topology of an APN and execute it by invoking services in the orchestration layer to establish end-to-end connections [21]. Once logical resources are created, they are stored both on the local computer running the GUI and a remote database for backup purpose. These objects not only serve as a basis for graphical display, but they are also used as resource identifiers when communicating with the service orchestration layer.

4.4. UCLPv2 Intra-domain Routing Capability

The objective of the routing function is to enable end-users to automatically establish end-to-end connections using logical resources belonging to a single organization. A two-page JFace routing wizard was implemented to provide a user interface consistent with the UCLPv2 GUI. A wizard is a JFace component that is capable of displaying multiple pages in a single dialog, and it is often used to collect information from the users.

A user can select two interfaces on source and destination switches in the resource list editor, and then launch the routing wizard. The first wizard page allows the user to

specify a desired bandwidth. Furthermore, the user has the option of finding a shortest path automatically or finding all the possible paths and then manually selecting one. In either case, a routing graph will be built from the logical resources of the organization. The vertices represent switches, and the edges represent available lightpaths. As a result of the bandwidth requirement specified by the user, only lightpaths with equal or greater bandwidth will be used in the construction of the graph.

The second wizard page displays the shortest path or all the possible paths depending on the user's choice. It also shows the corresponding number of hops taken for each path. Since the current UCLPv2 system does not assign a cost value to a lightpath, we have decided to use the number of hops as the cost value in the routing algorithm. When the routing wizard is finished, the connection management Web services provided by the UCLPv2 service orchestration layer will be invoked in order to establish an end-to-end connection.

When a user chooses the option of finding a shortest path, the program will execute the Dijkstra's algorithm. To find all the possible paths between two switches, the algorithm starts with the source vertex and discovers all its adjacent vertices. The discovered vertices are put into a first-in, first-out queue. The algorithm repeats the above steps with the first element of the queue as the new source until the queue is empty. It records the path information as it proceeds with the discovery of new vertices.

The following is the pseudo code of the algorithm.

```
vertex.setVisited(false) for all vertex;
FIFO_Queue.add(source);

while  FIFO_Queue ≠ Ø do
        u ← FIFO_Queue.removeHead
        for each adjacent vertex v of u do
                if v ≠ destination and v.getVisited() = false then
```

```
                    v.setVisited(true);
                    FIFO_Queue.add(v);
                end if
            end for
        end while
```

Chapter 5 - Design of Hierarchical Inter-domain Routing

UCLPv2 software provides a tool for users to provision and configure network resources within their own domain. By introducing the notion of lightpath, the heterogeneity of physical resources is well hidden from users. For example, a SONET channel, a WDM wavelength, or an MPLS tunnel can all be represented as lightpath objects. Another feature of the software is that it facilitates users to partition a lightpath into lightpaths of finer granularity to better suite the applications' bandwidth requirement. Since UCLPv2 is built upon a Web service orchestration layer, the system can be easily extended to include new Web services representing scientific instruments, network devices and so on.

Organizations within the context of UCLP are independent administrative domains. Currently organizational users can set up end-to-end connections using lightpaths belonging to their own domains. If the desired end-to-end connection consists of lightpaths that belong to different administrative domains, then the administrator of the organization must contact those organizations and acquire requested resources in an XML file through email. This is a manual process that is error prone and inefficient. Therefore, a useful extension to the software would be to add inter-domain routing capability; that is, the software will enable users to automatically acquire resources and establish end-to-end connections across multiple administrative domains.

Another motivation for our thesis work is that currently organizations within the context of UCLPv2 often have free resources to sublease to other organizations. To automate the resource acquirement process, two main problems need to be solved. The first problem is to develop a suitable resource advertisement mechanism. The second problem is to automatically acquire resources needed for establishing end-to-end connections by end-users. The framework will lead to the formation of a lightpath market place.

This thesis deals with the second problem. The goal is to develop new functionalities based on the current UCLPv2 system that facilitate users to establish end-to-end

connections using resources advertised by different organizations. To foresee the growth of the number of organizations participated in such a lightpath market place, the inter-domain routing solution must be scalable.

This chapter describes a conceptual approach to inter-domain routing for networks with condo-switches in Section 5.1. These ideas were proposed in the paper "Hierarchical inter-domain management for networks with condo-switches" [1]. Section 5.2 explains in details how the concepts are applied to the inter-domain routing design for UCLPv2.

5.1. Inter-domain Routing Concepts for Networks with Condo-switches

One salient characteristic of condominium networks is that organizations have shared ownership of resources, such as fiber lines and switches. Due to economical benefits, organizations prefer to share the capital cost and acquire partial ownership of resources. In practice, they can purchase different channels on the same fiber line. In addition, they can purchase different ports on the same optical switch. We categorize such type of switches as condo switches. Furthermore, these condominium networks are connected through condo switches. The advertised resources of an organization form a condominium network. Since each condominium network is independently managed by its owning organization, a distributed management function is required in order to establish inter-domain end-to-end connections [1].

A distributed network management mechanism is proposed by structuring networks into a hierarchy, where sub-networks are interconnected by condo switches. Furthermore, a hierarchical addressing scheme can be developed to facilitate the routing algorithm. "The multi-level hierarchical structure of networks and subnetworks provides an architecture for distributed processing of management functions that is very scalable." [1] The parent network knows the existence of its subnetworks, but has no knowledge of the internal structure of the subneworks. The approach is best illustrated by an example taken from [1].

Figure 16 shows an example of a physical configuration consisting of a number of switches (larger round circles) and terminal devices, such as computers (smaller round circles) interconnected by communication links. In particular, this configuration allows an end-to-end connection between the terminals H1 and H2 through the switches S1, S2, S3, S4, S5, S6, and S7. This figure does not show any administrative domains, although the different communication links and switches may belong to different owners.

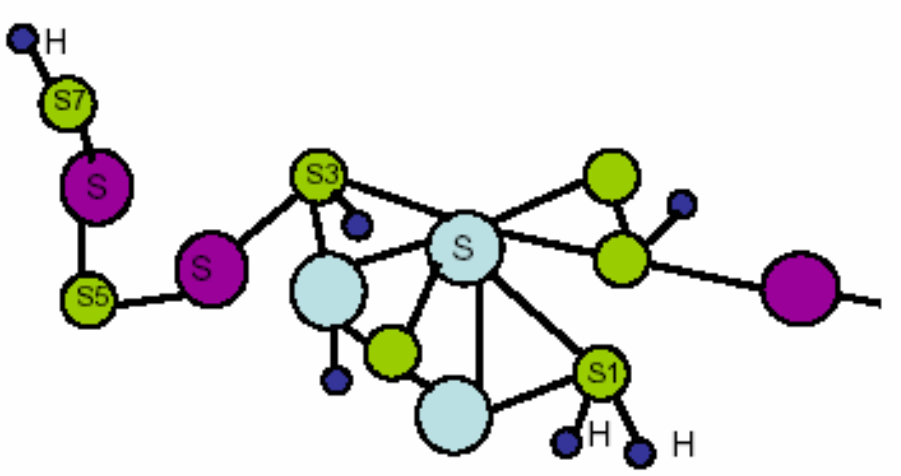

Figure 16: Example Configuration of Links, Switches, and Terminal Nodes [1]

Figure 17 shows the same configuration with the superposition of a hierarchical structure of administrative domains, in the following called "networks" or "subnetworks". For instance, network N6 consists of four subnetworks N2, N3, N4, and N5. These subnetworks do not contain any sub-subnetworks, except N2 which contains N1. The whole configuration shown in the figure is partitioned into four networks, N6, N7, N8 and N9. These networks are in fact subnetworks of the overall configuration, called network N10.

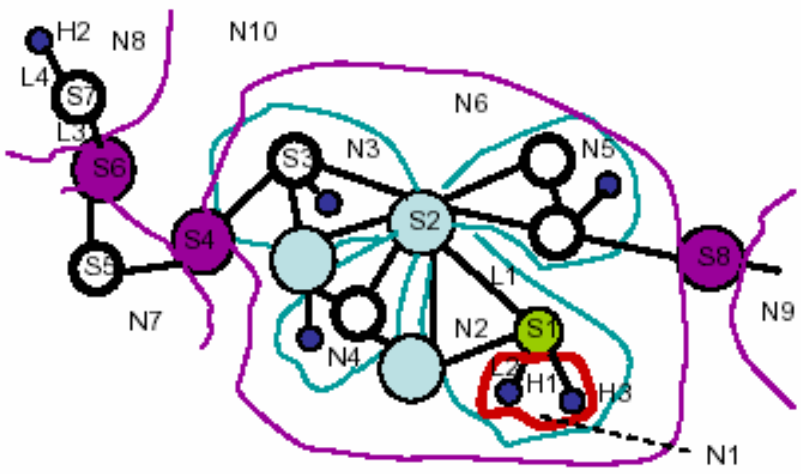

Figure 17: Hierarchical Inter-domain Structure Overlaid on Figure 16 [1]

We note that the colored switches in Figure 17 are condo switches, that is, for each of these switches, different ports belong to different networks. For instance, switch S1 has two ports belonging to network N1 (connected to the two terminals H1 and H3), and two ports that are connected to links that belong to network N2. Similarly, switch S2 is a condo-switch belonging to networks N2, N3, N4 and N5, while switch S4 interconnects network N6 (and the subnetwork N3) with network N7.

Given this hierarchical structure, the establishment of an end-to-end connection between two terminals will in general involve several networks. For the example of Figure 17, for instance, the establishment of an end-to-end connection between the terminals H1 and H2 may proceed as follows: Since the two terminals are located within the networks N6 and N8, respectively, the parent network of these two networks, network N10, will determine that the route should include a segment from switch S4 through network N7 to switch S6. The network N8 will determine the route from S6 to the terminal H2 which resides directly in this network, and network N6 would be responsible for finding a route from the terminal H1 to the switch S4. This will be done in two steps: first a segment from S2 (the chosen external condo-switch of the subnetwork N2 which contains H1) to S4, and then a segment from H1 to S2. The former segment can be obtained from the subnetwork

N3 by requesting a route between its external condo switches S2 and S4; and the latter segment will become the responsibility of the subnetwork N2.

To facilitate this kind of hierarchical routing, it is suggested to introduce a hierarchical naming scheme for networks, switches and terminals based on the hierarchical structure of the networks within which they reside. For instance, the addresses of the terminals H1 and H2 would be the following, assuming that N10 is at the highest hierarchical level: address of H1 = root/N10/N6/N2/N1/H1; address of H2 = root/N10/N8/H2.

A network can have the following three types of switches: external condo switches, internal condo switches, non-condo switches. The original definitions for external condo switches and internal condo switches are the following: "A switch S is an external condo switch of network N if at least one port of S is connected to N or a subnetwork of N, and at least one other port of S is connected to the parent network or another network N' or a subnetwork of N', where N' is a peer of network N in the network hierarchy. A switch S is an internal condo switch of network N if it is not an external switch of N, but it is an external switch of at least one of its subnetworks." [1] A complementary definition of non-condo switch is added to denote a switch whose ports are entirely owned by a single administrative domain. However, it is possible for a non-condo switch to become a condo switch when the ownership of some of its ports is transferred to other organization.

The logical resources of a network consist of switches, subnetworks, and lightpaths. A network provides a routing service to establish connections using its resources and/or the resources of its subnetworks. From the perspective of a single network, the following six types of connections can be established.

- A connection between two external condo switches of the network.
- A connection between two internal condo switches of the network, in other words, a connection between two subnetworks.
- A connection between an external condo switch and an internal condo switch.
- A connection between two non-condo switches of the network.

- A connection between a non-condo switch and an external condo switch.
- A connection between a non-condo switch and an internal condo switch.

5.2. Inter-domain Routing Design for UCLPv2

The current UCLPv2 software enables administrators to group logical resources including lightpaths and interfaces, into a resource list XML file. Therefore, when an organization has free resources for the use of others, an administrator of the organization can use the existing tool to create a resource list which contains those free resources. Figure 18 shows an UML diagram describing the current relationship among resources in such a resource list.

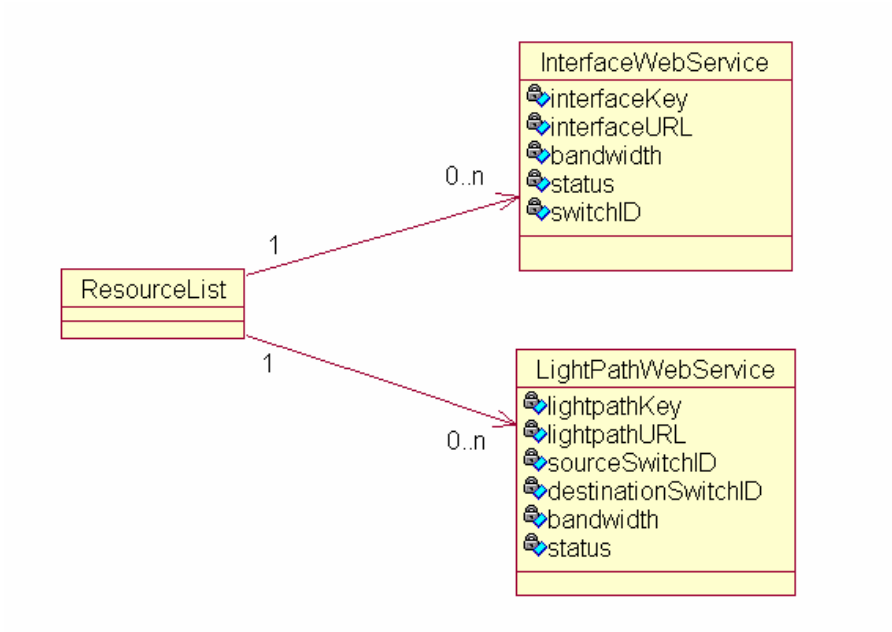

Figure 18: UML Diagram of the Current UCLPv2 Resource List

An assumption made is that all the lightpaths and interfaces in a resource list could be connected through switches. Consequently, these resources essentially form a connected network. Every organization using UCLPv2 could have such a resource list; as a result, there would be many such networks. Based on the fact that the networks of these organizations are connected through condo switches, we can apply the hierarchical inter-

domain routing concepts to UCLPv2. By establishing the multi-level hierarchy among networks owned by different organizations, we have a framework for automatically finding an end-to-end connection across multiple administrative domains. This may lead, in the future, to the establishment of a market for lightpath resources.

In order to realize this collaboration between the different UCLP organizations, we propose the introduction of an additional network management service to be provided by each participating organization. The purpose of this network management Web service is to provide routing capabilities based on an organization's free resources.

5.2.1. Resource List Modification

A resource list XML file created by an administrator, as defined by the UCLP system, must be modified to include additional information in order to maintain the hierarchical relationship among networks. The modification to the resource list is illustrated in Figure 19. Since a resource list is an XML file, the UML diagram is used here to represent relationships among the XML elements.

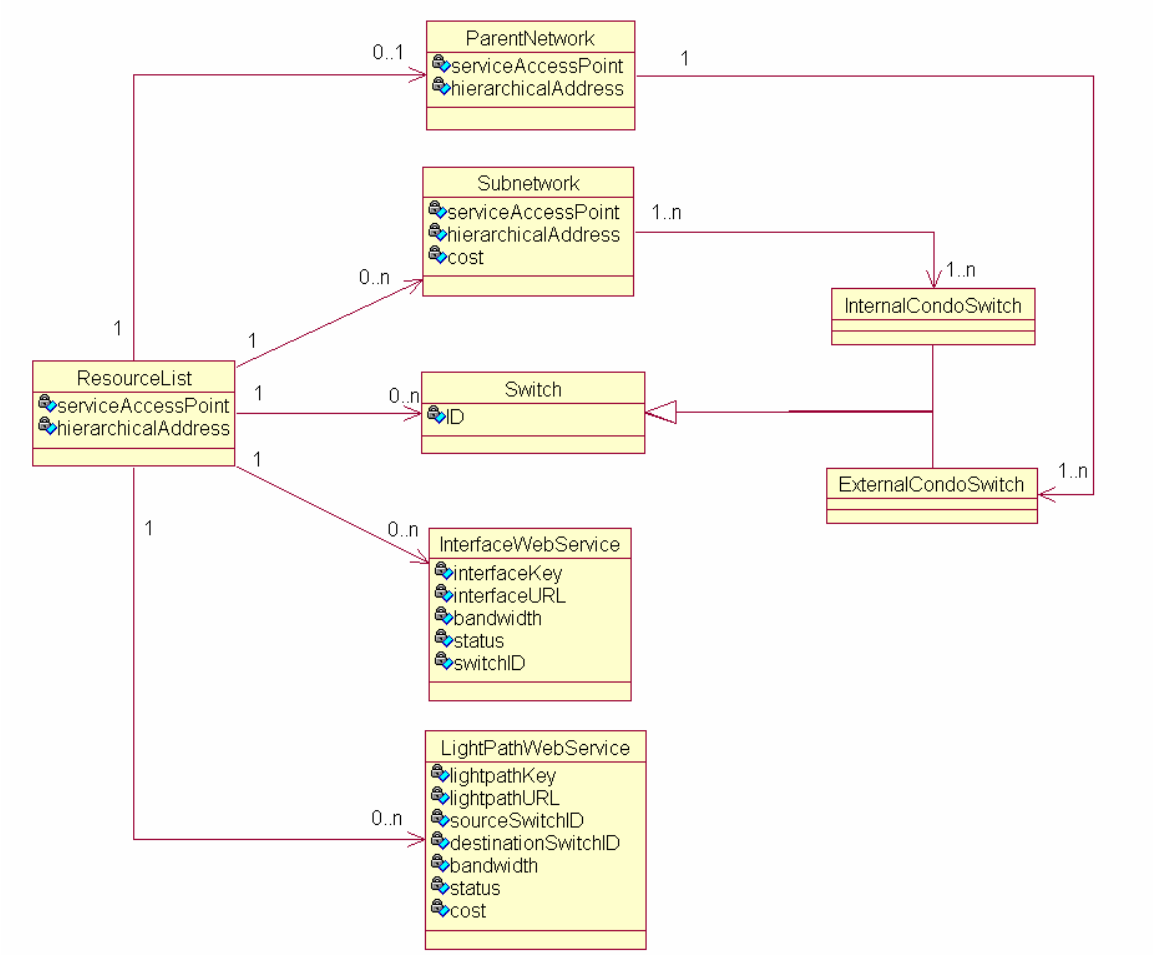

Figure 19: UML Diagram of a Modified Resource List

The diagram shows that a network can have at most one parent network and many subnetworks. The network is connected with its parent network through at least one external condo switch. Furthermore, the network is connected with any of its subnetwork through at least one internal condo switch. The information is maintained during the network registration process as explained in Section 5.2.3.1.

A resource list file also provides topological information and the provisioned cost values associated with the resources in order to facilitate the routing functions. Each network management entity has a service access point, namely a URL for the Web service operations it provides. The service access points of the current network, the subnetworks,

and the parent network are stored as attributes of the XML elements, in addition to the hierarchical addresses of the networks.

To provide resources for third-party usage, an organization will store the XML file at a predefined directory within the Web service container. The file will be read during the network management Web service initialization phase and maintained subsequently.

5.2.2. Design Choices for Hierarchical Addressing

In order to facilitate routing, a hierarchical addressing scheme was designed to uniquely identify network elements within the hierarchy. The address of a sub-network is the concatenation of the address of its parent network and the name of the sub-network. The name of a subnetwork must be unique within the scope of the parent network.

There are three design choices. The first choice is to allow the parent network to assign a unique name to a subnetwork when the subnetwork registers with the parent network. The second choice is to allow the subnetwork to propose a name during the registration process. The parent network checks the uniqueness of the proposed name. The third choice is to use the service access point of the network management Web service as the unique name of a network. However, the last approach is impractical to adopt because a hierarchical address made up of URLs could become very lengthy. We have decided to choose the second solution because it gives a subnetwork the flexibility to choose a proper name.

5.2.3. Network Management Web Service

The network management Web service of an organization has the following five operations: registering a subnetwork, unregistering a subnetwork, updating switch resources, fulfilling a routing request by establishing an end-to-end connection, and releasing reserved resources. The network management Web services of different organizations collaborate to form a hierarchical relationship among networks and to

facilitate the inter-domain routing process. Communication takes place only between management Web services having parent-child relationship in the hierarchy.

5.2.3.1. Subnetwork Registration

Every network management service provides a function to register subnetworks. The function is crucial for the formation of a hierarchy of networks. In order to form a proper parent-child network relationship, the following condition must be satisfied: the child network must be connected with the parent network through at least one condo switch. This condition ensures contiguousness of the networks, and it is checked when a child network registers with a parent network. The name of the subnetwork must be unique within the scope of the parent network.

During the initialization phase of any network management Web service, the service will invoke the registration operation of its parent network. The input to this function consists of four parameters. The child network must provide a proposed name, a list of all its external condo switches, its SAP (URL), and an estimated cost value for using the resources in the subnetwork. The parent network compares the external condo switches of the subnetwork with all of its own switches. If at least one match is found, then the network contiguousness condition is satisfied. In addition, the parent network compares the proposed subnetwork name with the names of its other subnetworks to guarantee uniqueness. When both conditions are met, the parent network will update its resource list to include the subnetwork. The parent network will construct a hierarchical address and return it to the subnetwork.

There is a general case regarding switches to be considered when a subnetwork registers. This happens when some of the external switches of the subnetwork do not exist in the resource list of the parent network. The parent network updates its resource list by adding the subnetwork and the new switches. These new switches must be added to the parent network as its external switches. As a result, the parent network must inform its own parent network of the addition of these new switches. The purpose is to propagate the

connectivity change through the hierarchy. This is achieved through another web service operation called updateSwitchResources.

A special case occurs when all the external switches of the subnetwork already exist in the resource list of the parent network. The parent network updates its resource list by only adding the subnetwork.

5.2.3.2. Switch Resource Update

The objective of the operation is to provide a mechanism for a child network to notify its parent network about the addition or removal of new external condo switches. A child network invokes the operation by providing a list of condo switches that have been added to or removed from its resource list. As a result, the parent network will update its resource list, specifically adding external condo switches to or removing them from the child network. If any condo switch does not already exist in the parent network, then the parent network is responsible for informing its parent network about the addition of the new condo switches. In that case, the changes will propagate up the hierarchy if necessary.

5.2.3.3 Subnetwork Unregistration

The purpose of the operation is to allow a subnetwork to leave a parent network. The input to this function is the hierarchical address of the subnetwork. The parent network will update its resource list to remove information relevant to the subnetwork. In case that the external condo switches of the parent network are affected, then it must inform its parent network by invoking the updateSwitchResources operation.

5.2.3.4 Establishing End-to-End Connections

The Web service routing operation is achieved through the following three steps: the analysis of the input hierarchical addresses of the source and destination switches, the search for a least-cost path, and the establishment of the path (that is, making cross-connections on the switches en route). The input parameters of the operation are the hierarchical addresses of the source and destination switches, and a desired bandwidth. The output of the operation is a list of identifiers of the lightpaths and associated networks used to establish the route.

The purpose of the analysis of the input addresses is to determine whether the current network is the lowest common parent network of the source and destination switches. For example, if the input addresses are root/N1/N2/S1 and root/N1/S2, but the current network is N2 or root, then the current network is not the lowest common parent network of the two switches, which is N1. Consequently, the routing request will be delegated to the routing Web service of network N1. The address analysis process is executed as the first step of the routing Web service. It ensures that the process of searching and establishing a path starts at the lowest common parent network of the source and destination switches.

By representing a network including its subnetworks by a hypergraph, the program can execute the Dijkstra's algorithm in order to search for a least-cost path. The searching algorithm uses a single routing metric to calculate total cost of a path. A scalar value must be assigned to every logical link and subnetwork by the network administrators. In a lightpath market the metric usually represents the monetary cost of using resources. In addition, the cost assigned to a subnetwork can represent the average cost of using resources in the subnetwork. The drawback of assigning a single value to a subnetwork is that the cost of a subnetwork becomes independent of the length of the connection through that subnetwork. This imposes a limitation on the system. Since the metric is simply a number in the system, administrators have the option to use the metric to represent physical length of a fiber link, propagation delay, link reliability and so forth. However, the choice of the metric must be consistent throughout the system so that the total cost of a path is meaningful.

The establishment of a path is based on the connection management Web services provided by the UCLPv2 service orchestration layer. By invoking the connection management Web service with a list of lightpath identifiers, the system can automatically construct an end-to-end path.

A natural way to represent the hierarchical networks is to use hypergraph. A hypergraph is a graph that contains at least one hyperedge. A hyperedge is defined as an edge connecting possibly more than two vertices [17]. A physical network has three categories of resources: switches, lightpaths, and subnetworks. Switches can be represented by vertices. Lightpaths can be represented by regular edges. The challenge is how to represent a subnetwork. A subnetwork is connected to its parent network through its external condo switches, and these switches are already represented in the parent network as vertices. Furthermore, we made the assumption that the subnetwork is fully connected, that is, it may establish a connection between any two of its external switches. Based on this assumption, the subnetwork can be represented by a hyperedge connecting all its external switches. Consequently, a physical network can be transformed into a hypergraph as shown in Figure 20, where the rectangles are the hyperedges.

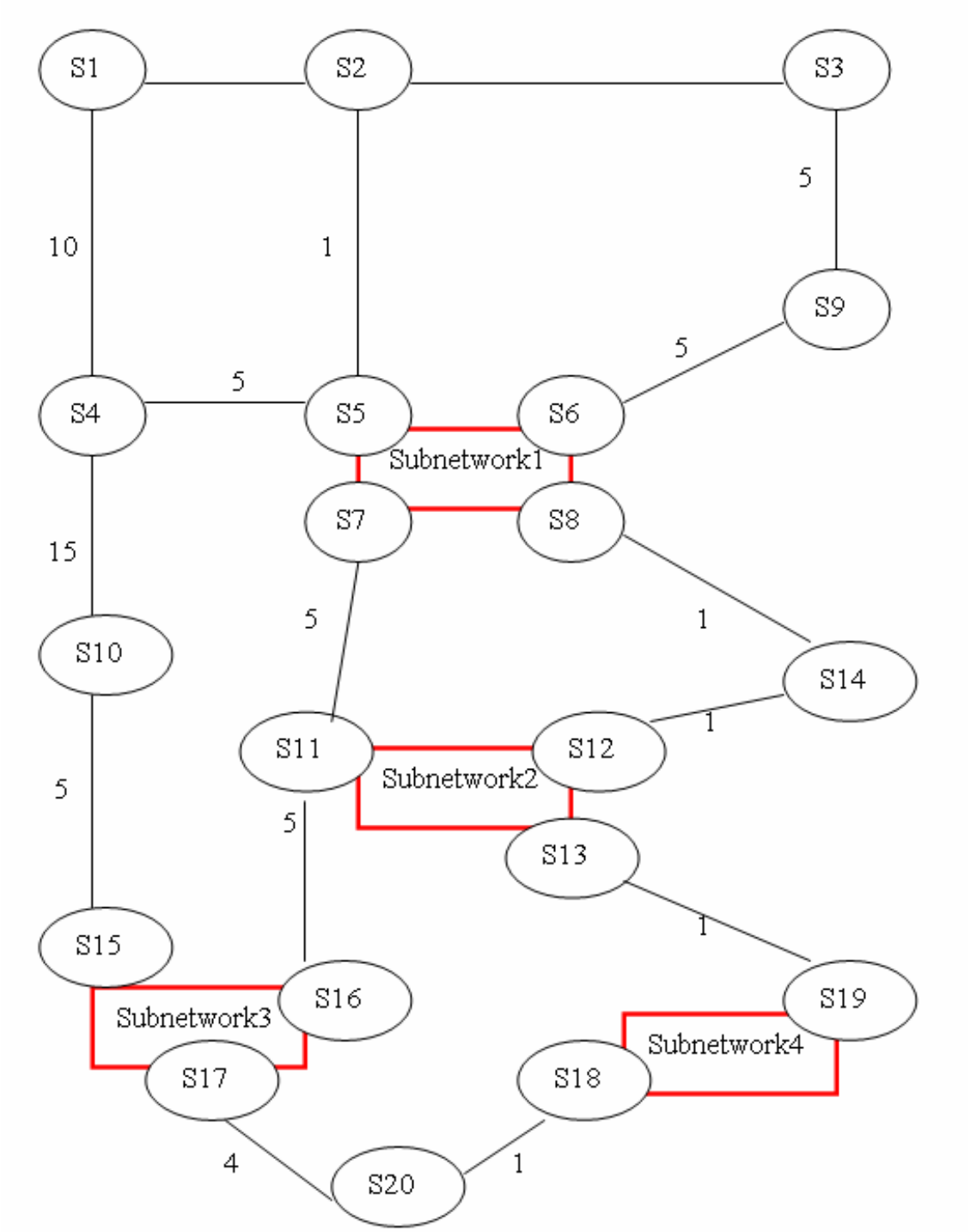

Figure 20: Hypergraph Representation of a Physical Network

After a hypergraph representing the network topology is constructed from the list of resources, the Dijkstra's algorithm can be adapted to find the shortest path on the hypergraph. When the Dijkstra's algorithm is applied to a regular graph, the edge

relaxation operation is performed on one adjacent vertex connected by an edge. To extend the algorithm to a hypergraph, we can perform the edge relaxation operation on all the adjacent vertices connected by a hyperedge. The implementation of the algorithm is described in Section 6.2.3.

In order to find a least-cost path in the hypergraph, we must assign cost values to edges or hyperedges. An edge representing a lightpath will inherit the cost value associated with the lightpath. The implication is that if two lightpaths having the same source and destination switches, but different cost values, will be represented by two different edges. An hyperedge representing a subnetwork will inherit the cost value associated with the subnetwork.

5.2.3.5. Release Resource

The purpose of the operation is to decompose an end-to-end connection and release its resources. Decomposing an end-to-end connection is achieved by invoking the connection management Web service of UCLPv2 to undo cross connections on the lightpaths. Releasing resources is done by updating the status of the lightpaths in the resource list to become available. The input parameter of this operation is a list of lightpath identifiers. The output parameter is a boolean value indicating whether the operation was successful.

Chapter 6 - Implementation of Hierarchical Inter-domain Routing

The implementation of the distributed hierarchical management system is based on Web services. The implementation consists of two modules: a client program and a network management Web service. This section briefly describes the client program and focuses on the implementation of the most important network management Web service operation, namely the establishment of an end-to-end connection. Furthermore, an example usage of the network management Web service was designed to validate the implementation of the system.

6.1. Client Program

A client program provides a graphical user interface for end-users to request end-to-end connections. It is implemented as an Eclipse Jface wizard dialogue and integrated into the UCLPv2 GUI. A user provides the hierarchical addresses of the source and destination network elements, and the URL of the network management Web service representing the local network where the user is known as a legitimate user. Furthermore, a user specifies a desired bandwidth. The client program invokes the routing Web service to establish an end-to-end connection. Another administrator client program is implemented to provide a graphical user interface for invoking the following management functions: subnetwork registration and subnetwork unregistration. This is important for the establishment of the network management hierarchy.

6.2. Implementation of the END-TO-END Connection Establishment Operation

The operation is capable of establishing both intra-domain and inter-domain end-to-end connections. The process starts with the analysis of input addresses. The implementation

of the search for a least-cost path deals with three scenarios. The establishment of the path involves the invocation of the UCLPv2 connection management Web service.

6.2.1. Analysis of Source and Destination Addresses

The objective is to determine if the current network is the proper place in the hierarchy to process the routing request. The process starts with finding the lowest common parent network of the two switches first, and then compares it with the current network to determine its relative position in the hierarchy. There are three possible situations. The common parent network could be located in a higher position in the hierarchy, or a lower position in the hierarchy, or in a peering position. The implementation of the analysis process for each situation is best illustrated with the following examples. Figure 21 shows the hierarchical relationship between network management entities.

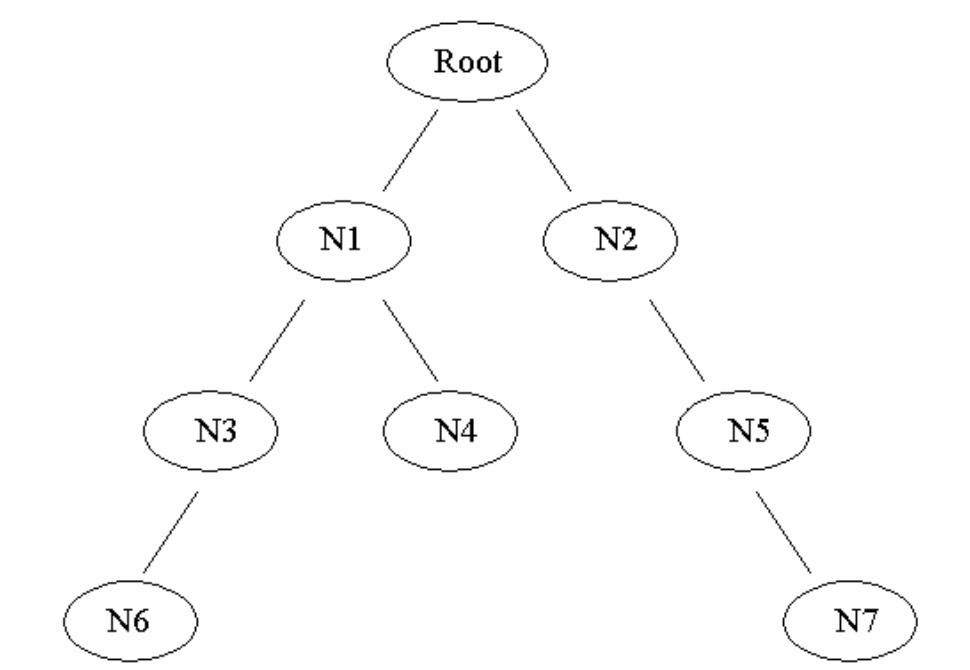

Figure 21: Hierarchical Relationship between Network Management Entities

Network N1 stores the following information in the resource list:

- Its own hierarchical address, namely root/N1
- Its own URL, for example, http://www.networkN1.com/webservices

- URL of its parent network, for example, http://www.networkroot.com/webservices
- Hierarchical addresses of its child networks, namely root/N1/N3 and root/N1/N4
- The URLs of its child networks, for example, http://www.networkN3.com/webservices and http://www.networkN4.com/webservices

Other networks store information in a similar fashion.

An example where the common parent network is located in a higher position than the current network would be that a client program sends a routing request to network management entity of N6 to establish a path between two switches in network N6 and N7. The input source and destination addresses are root/N1/N3/N6/S1 and root/N2/N5/N7/S2.

Network N6 scans the two addresses from left to right, and extracts the common portion of the addresses. In this case, the common address is root. It then compares the common address with its own hierarchical address. If the comparison indicates that the common address belongs to a network with a higher position in the hierarchy, it will delegate the routing request to its parent network, namely N3. The same analysis process takes place at N3. Eventually, the root network receives the request and launches the searching process for a least-cost path between S1 and S2.

An example where the common parent network is located in a lower position than the current network would be that a client program sends a routing request to network management entity of the root network to establish a path between two switches in network N3 and N4. The input source and destination addresses are root/N1/N3/S3 and root/N1/N4/S4. The root network scans the two input addresses and determines that root/N1 is the common portion. By comparing the common address with its own address, it concludes that the common address belongs to a network with a lower position in the hierarchy. Furthermore, it can conclude that one of its child networks is the proper network to delegate the routing request to. In this case, that network is N1. Since N1 is

the least-common parent network of the S3 and S4, it executes the routing request and returns the result to root, which returns the result to the client program.

An example where the common parent network is located in a peering position would be that a client program sends a routing request to network management entity of N2 to establish a path between two switches in network N3 and N4. The input source and destination addresses are root/N1/N3/S3 and root/N1/N4/S4. The common address is root/N1. The process continues by comparing the common address with the hierarchical address of the current network, which is root/N2. The comparison indicates that these two networks are peers, and the root network is their common parent network. Therefore, the process will delegate the routing request to the root network, which will determine that N1 is the proper network to handle the routing request.

6.2.2. Establishment of Three Types of Connections

In Chapter 5 we concluded that there are six possible types of connections that can be established. Instead of providing six different operations for finding different types of connections, we decided to implement a single operation that is capable of finding a connection in all these cases. The advantage of this approach is to simplify client-side processing. Client programs need not to distinguish the type of a connection and find the corresponding operation to invoke. Instead, client programs simply input source and destination parameters when invoking the route establishment operation.

From an implementation perspective, these connections can be further categorized into three types, depending on the nature of input source and destination addresses. Type 1 occurs when both source and destination switches belong to the current network. For example, network N1 in Figure 21 receives a routing request with the following input parameters: root/N1/S1 and root/N1/S2. If an end-to-end connection between S1 and S2 can be constructed using resources only from network N1, then this scenario is equivalent to the intra-domain routing. Type 2 occurs when the source switch is located in the current network, but the destination switch is located in one of its subnetworks, or vice

versa. An example of this scenario would be that network N1 receives a routing request with the following input parameters: root/N1/S1 and root/N1/N3/S2. Type 3 occurs when both source and destination switches belong to two of its subnetworks, respectively. An example of this scenario would be that network N1 receives a routing request with the following input parameters: root/N1/N4/S1 and root/N1/N3/S2.

6.2.2.1. Establishing Connection of Type One

The case implies that both switches are resources belonging to the current network. The program executes the routing algorithm and finds a least-cost path if it exists. The routing algorithm stops as soon as the destination is reached. If the resultant path does not contain any subnetwork, then the program simply invokes the connection management Web service to concatenate the lightpaths. On the other hand, if the path contains subnetworks, then the program invokes the routing Web services of these subnetworks to find segments of the path. If one of the subnetworks fails to find the segment, then this entire path must be abandoned. An implementation decision is made in order to simplify the handling of subnetwork failure; that is, if a subnetwork fails to establish a path between two of its external condo switches, then it will be removed from the graph before the re-execution of the routing algorithm. As a result, the routing graph is updated, and the above routing process will be repeated.

6.2.2.2. Establishing Connection of Type Two

In this case one of the endpoints is located in a subnetwork. Once a path from the source switch to the subnetwork is found, the program delegates the request to the subnetwork to find the remaining segment of the path.

Since a subnetwork could have more than one external condo switch, this case requires the routing algorithm to find a list of shortest paths from a single source switch to multiple external switches of the subnetwork. The program will compare all the resultant

paths by their total cost, and choose a path with the least cost. Since the path chosen is connected with the subnetwork through a particular external switch, the program will send a routing request to the subnetwork in order to find a segment from that external switch to the destination switch.

In case a segment cannot be found by the subnetwork, the program will use the next shortest path to the subnetwork and send a routing request with a different external switch. The process repeat until a path can be successful established or the list of paths is exhausted.

6.2.2.3. Establishing Connection of Type Three

In this case the two endpoints are located in different subnetworks. The program must find a shortest path connecting the two subnetworks, then delegate the requests to both subnetworks in order to find the two remaining segments of the end-to-end path.

Since either subnetwork can have multiple external condo switches, this demands the execution of the routing algorithm multiple times in order to find all shortest paths from every external condo switch of the source subnetwork to every external condo switch of the destination subnetwork. Since these are the shortest paths between every two external switches of the subnetworks, the program performs similar steps as described in scenario two to determine the absolute shortest path between the two subnetworks. Subsequently, the program sends routing requests to the subnetworks to establish the end-to-end connection. The failure handling steps are similar to scenario two.

6.2.3. Implementation of Shortest-Path Routing Algorithm

The Java implementation of the hypergraph data structure is used to facilitate the implementation of the shortest-path routing algorithm.

6.2.3.1. HyperGraph Data Structure

Java classes are created to implement the hypergraph data structure, and their UML class diagram is shown in Figure 22. Class Switch and LightpathWebService are existing class definitions of UCLPv2. They are shown here to reveal their relationships with the hypergraph.

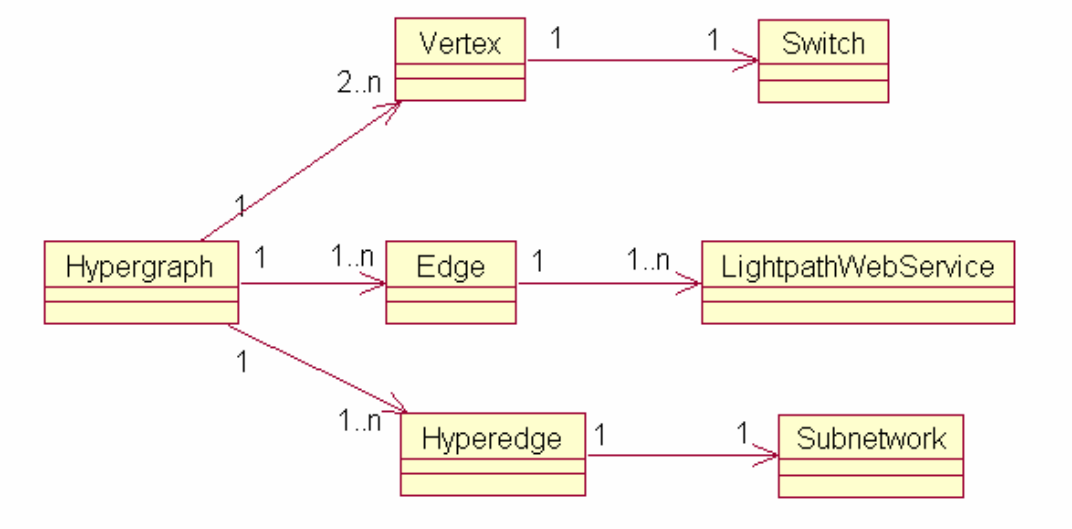

Figure 22: UML Class Diagram of Graph Data Structure

- Class Hypergraph: The implementation of the hypergraph is based on an adjacent list. The hypergraph uses a Java HashMap container to maintain its vertices, edges, hyperedges and their relationships. The HashMap uses a Vertex object as its key, and a list of adjacent edges and/or hyperedges as its value.
- Class Edge: An instance of an edge contains a list of lightpaths. These lightpaths share the same cost, and the same source and destination pairs. It also contains a cost value, source and destination vertices. The implication is that if two lightpaths with the same source and destination pairs, but different cost values will be represented by two different edges.
- Class Vertex: Vertex contains the corresponding switch information.
- Class Hyperedge: Hyperedge contains a list of vertices, a cost value, and a reference to a subnetwork.

6.2.3.2. Routing Algorithm

The following five variables represent different data structures that are used to implement the routing algorithm. These variables are referred to in the pseudo codes below as unResolvedVertices, resolvedVertices, precedingVertices, precedingEdges, costMap, respectively.

- unResolvedVertices: A Java PriorityQueue structure is used to store unresolved vertices. The PriorityQueue stores elements in a specific order default to ** not clear ascending order. We set up the queue so that the vertices are ordered according to their cost values, with the head of the queue being the element with the minimum cost. The use of this structure saves coding effort when the algorithm retrieves the vertex with minimum cost from the queue.

- resolvedVertices: A Java HashSet structure is used to store resolved vertices.

- precedingVertices: A Java HashMap structure is used to store preceding vertices. The key stores the current vertex, the value is its preceding vertex.

- precedingEdges: A Java HashMap structure is used to store preceding edges. The key stores the current vertex, the value is its preceding edge.

- costMap: A Java HashMap structure is used to store cost values. The key stores a vertex, the value is the total cost from the source to the vertex.

The pseudo code below represents a modification of Dijkstra's algorithm in order to allow for the processing of hyperedges. It also checks whether an edge contains at least

one available lightpath and the associated bandwidth value is equal or greater than the input bandwidth before proceding with the edge relaxation operation.

```
costMap.add(s, 0), where s is the source node
costMap.add(u, ∞) for all u ≠ s
resolvedVertices ← Ø
unResolvedVertices ← all vertices
while unResolvedVertices ≠ Ø do
        u ← Extract-Minimum(unResolvedVertices)
        if u = destination then
                exit the algorithm
        end if
        resolvedVertices.add(u)
        for each adjacent object of u do
                if the object is a regular edge then
                        if the edge contains at least one available lightpath and
                           the lightpath's bandwidth >= input bandwidth then
                                v ← adjacent vertex of u connected by the edge
                                relaxEdge(v,u)
                        end if
                else if the object is a hyperedge then
                        for each vertex v connected by the hyperedge do
                                if v ≠ u then
                                        relaxEdge(v,u)
                                end if
                        end for
                end if
        end for
end while
```

Relax Edge Function:

```
Input: v1, v2
        if resolvedVertices contains v1, then return
        c1 ← costMap.getCost(v1)
        c2 ← costMap.getCost(v2)
        c3 ← cost of the edge between v1 and v2
        if c2+c3 < c1 then
                costMap.update(v1,c2+c3)
                precedingVertices.update(v1,v2)
        end if
```

When the algorithm processes an hyperedge, it does not check for the bandwidth requirement. The assumption made at this point of the algorithm is that the subnetwork represented by the hyperedge could establish a path satisfying the bandwidth requirement. When the parent network invokes the routing function of the subnetwork, it will input the bandwidth parameter in addition to the source and destination addresses. If the subnetwork cannot find a path satisfying the bandwidth requirement or for any other reason, the subnetwork will notify the parent network about the failure of the routing request and the parent network will try another path as explained in Section 6.2.2.1, 6.2.2.2, and 6.2.2.3.

Since the precedingVertices variable is used to keep track of the shortest path, it must be updated when the edge relaxation operation results in a new preceding vertex.

6.3. Class-Level Implementation of the Hierarchical Network Management Web Service

Figure 23 shows an UML class diagram that illustrates the relationships among the main classes implemented.

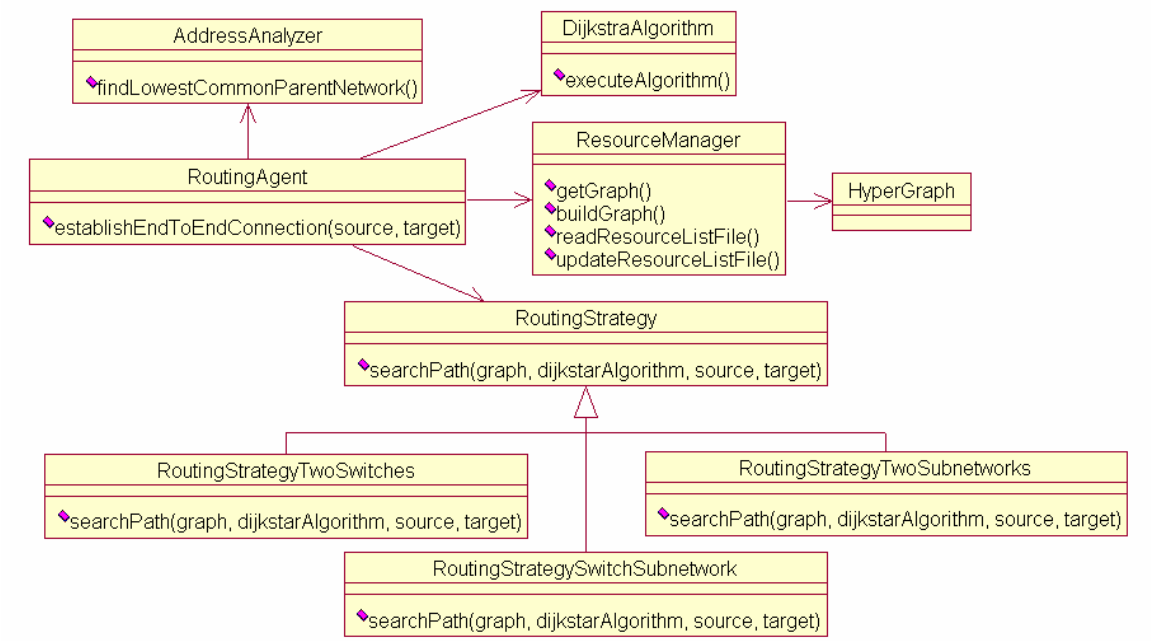

Figure 23: UML Class Diagram of the Implementation of the Network Management Service

- The RoutingAgent class provides the end-to-end connection establishment operation. It implements high-level logic, but delegates specific operations to other classes to which it holds object references.
- The RoutingStrategy class has three subclasses that deal with the three different types of end-to-end connections.
- The ResourceManager class is implemented to read from and write to the XML file. It uses a third-party tool, Apache Xerces, to parse the XML file.
- The Dijkstra Algorithm class encapsulates the implementation details of the application of the Dijkstra's algorithm to a hypergraph.
- The AddressAnalyzer class provides an operation to determine the lowest common parent network of the source and destination.

An organization stores the resource list XML file at a predefined directory in the Web server. An example resource list XML file is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
<resource-list>

        <hierarchical-address>root/N1</hierarchical-address>
        <service-access-point>
                http://www.networkN1.com/routingservice
```

```
</service-access-point>

<parent-network>
        <service-access-point>
                http://www.rootnetwork.com/routingservice
        </service-access-point>
        <hierarchical-address>root</hierarchical-address>
        <external-condo-switch>cdn-switch1</external-condo-switch>
        <external-condo-switch>cdn-switch2</external-condo-switch>
</parent-network>

<subnewtwork>
        <service-access-point>
                http://www.networkN2.com/routingservice
        </service-access-point>
        <hierarchical-address>root/N1/N2</hierarchical-address>
        <internal-condo-switch>ons-switch1</internal-condo-switch>
        <internal-condo-switch>ons-switch2</internal-condo-switch>
        <cost>20</cost>
</subnewtwork>
<subnewtwork>
        <service-access-point>
                http://www.networkN3.com/routingservice
        </service-access-point>
        <hierarchical-address>root/N1/N3</hierarchical-address>
        <internal-condo-switch>ons-switch1</internal-condo-switch>
        <internal-condo-switch>ons-switch2</internal-condo-switch>
        <cost>10</cost>
</subnewtwork>

<lightpath-webservice>
        <lightpath-url>http://www.canarie.com/LP-WS1</lightpath-url>
        <lightpath-key>LPkey1</lightpath-key>
        <source-switch id="cdn-switch1" port="1000">
        </source-switch>
        <target-switch id="cdn-switch4" port="2000">
        </target-switch>
        <bandwidth>51 mbps</bandwidth>
        <cost>10</cost>
        <status>available</status>
</lightpath-webservice>
<lightpath-webservice>
        <lightpath-url>http://www.canarie.com/LP-WS2</lightpath-url>
        <lightpath-key>LPkey2</lightpath-key>
        <source-switch id="cdn-switch3" port="100">
        </source-switch>
```

```
            <target-switch id="cdn-switch2" port="200">
            </target-switch>
            <bandwidth>155 mbps</bandwidth>
            <cost>20</cost>
            <status>available</status>
        </lightpath-webservice>

        <interface-webservice>
            <interface-url>http://www.canarie.com/I-WS1</interface-url>
            <interface-key>interfaceKey1</interface-key>
            <switch id="cdn-switch1" port="500">
            </switch>
            <bandwidth>155 mbps</bandwidth>
            <status>available</status>
        </interface-webservice>
        <interface-webservice>
            <interface-url>http://www.canarie.com/I-WS2</interface-url>
            <interface-key>interfaceKey2</interface-key>
            <switch id="cdn-switch5" port="500">
            </switch>
            <bandwidth>51 mbps</bandwidth>
            <status>available</status>
        </interface-webservice>

</resource-list>
```

The Eclipse Web Tools Platform (WTP) was chosen as the integrated development tool. The WTP extends the functionalities of Eclipse 3.1 platform to support the development of J2EE applications including Web services. WTP enables developers to create a Web service from a WSDL file, or convert an existing Java class into a Web service. It also offers tools such as the Web service explorer and SOAP message monitor for testing purpose. In addition, the latest Java JDK version 1.5 was used.

The network management Web service was deployed in the Apache Axis SOAP engine version 1.3, which requires the Apache Tomcat as its Web server. UCLPv2 software was also installed according to the user manual [21]. A Java-based build tool, Apache Ant, was used to write service startup scripts.

6.4. Consideration of Resource Access Rights

The current UCLPv2 system associates access rights with resources such as lightpaths, interfaces, and APNs. An organization is the owner of its resources. Furthermore, the access right policies specify that only the current owner of a resource has the rights to perform operations on the resource. However, the ownership of a resource can be transferred from one organization to another. The GUI provides a resource exportation function which effectively transfers the ownership of the resources. The ownership information is maintained at the network element Web services in the form of a stack, with the top of the stack being the current owner, and the bottom of the stack being the original owner. Consequently, the ownership information will be verified before performing an operation on the network element. For example, to perform an operation such as concatenating two lightpaths (making a cross connection) at a common switch, the network element Web service must first verify that both lightpaths are owned by the same organization [20].

However, this access right mechanism imposes a problem for the inter-domain routing. The problem manifests when concatenating two lightpaths that belonging to two different organizations, representing a parent network and a subnetwork. To establish an end-to-end path involving a subnetwork, the parent network invokes the routing Web service of the subnetwork to establish a segment of the path. The subnetwork executes its routing process and constructs a new lightpath that represents the segment of the path. The subnetwork returns the lightpath to the parent network. However, since the owner of the lightpath is still the subnetwork, the parent network cannot concatenate the returned lightpath with any of its own lightpahs at a condo switch.

There are two solutions to the above problem. The first solution requires that when an organization advertises resources for the use of others, the advertisement mechanism must disassociate the access rights with these resources to allow any organization to use it. The second solution is to use the current resource exportation function to change the access rights associated with the resources as follows. When the routing Web service of a

subnetwork establishes a path segment upon the request of its parent network, it must export the lightpath, representing the segment, to its parent network. The exportation operation will change the owner of the lightpath to the parent network, thereby, enabling the parent network to perform operations on the lightpath.

6.5. Network Management Web Service Example Usage

The following use case of the network management Web service was designed to demonstrate the successful establishment of end-to-end connections. There are six networks in the example forming a hierarchical management system, as shown in Figure 24.

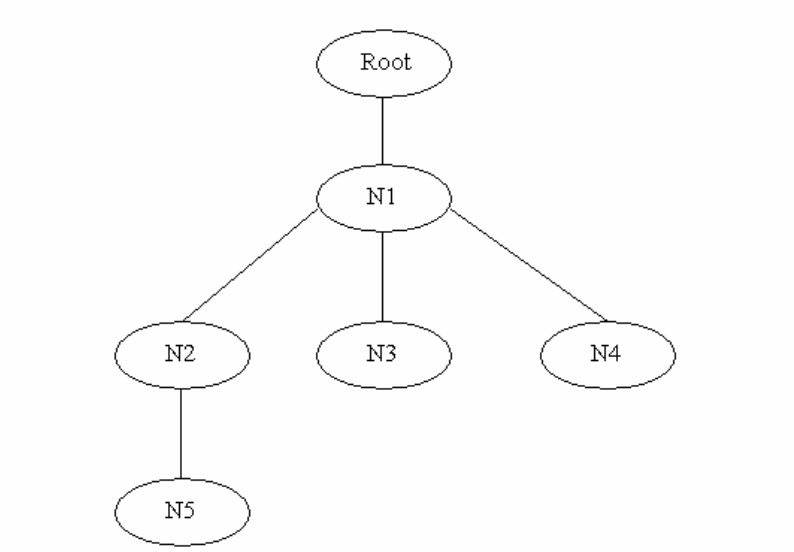

Figure 24: an Example of a Hierarchical Network Management System

In order to form the hierarchy, the Web service of the root network is launched first, and network N1 registers with it as a child network. The name of a network can be arbitrarily chosen, we are using N1 for convenience. Next, network N2 invokes the subnetwork registration operation of network N1 with the proposed network name and its external condo switches. Similar registration processes take place with N3, N4, and N5.

A client program sends a routing request to its local network which is N5. The source address is Root/N1/N2/N5/S1, and the destination address is Root/N1/N4/S2.

The topologies of network N1, N2, N3, N4, N5 are shown in Figure 25, 26, 27, 28, 29 respectively. These topologies are designed to demonstrate the routing of the three types of connection segments. Network N1 establishes a segment between two subnetworks, namely N2 and N4. Network N2 establishes a segment between one of its external condo switch and its subnetwork N5. Network N5 establishes a segment between one of its external condo switches and the source switch.

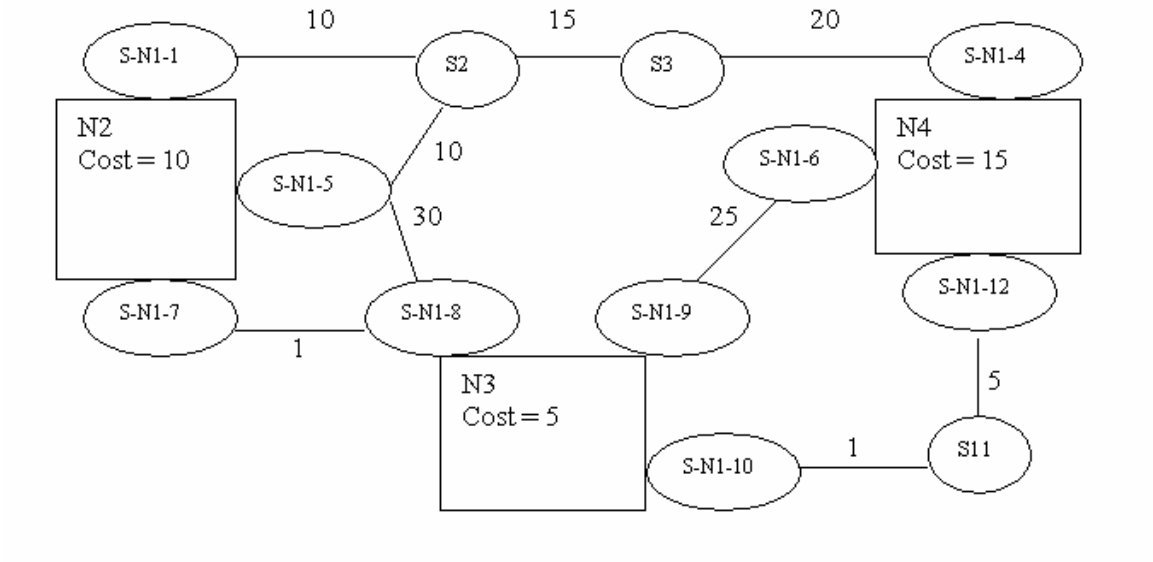

Figure 25: Topology of Network N1

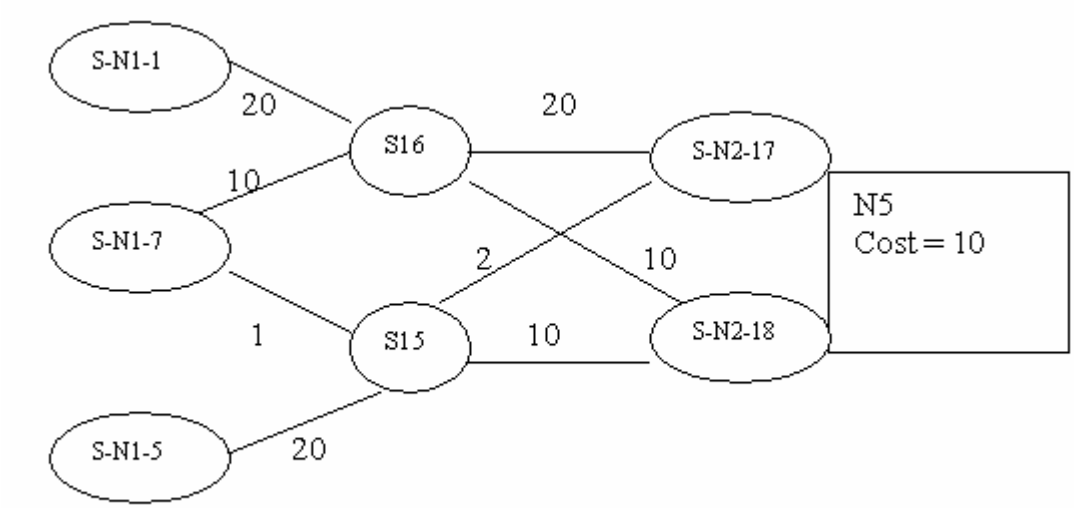

Figure 26: Topology of Network N2

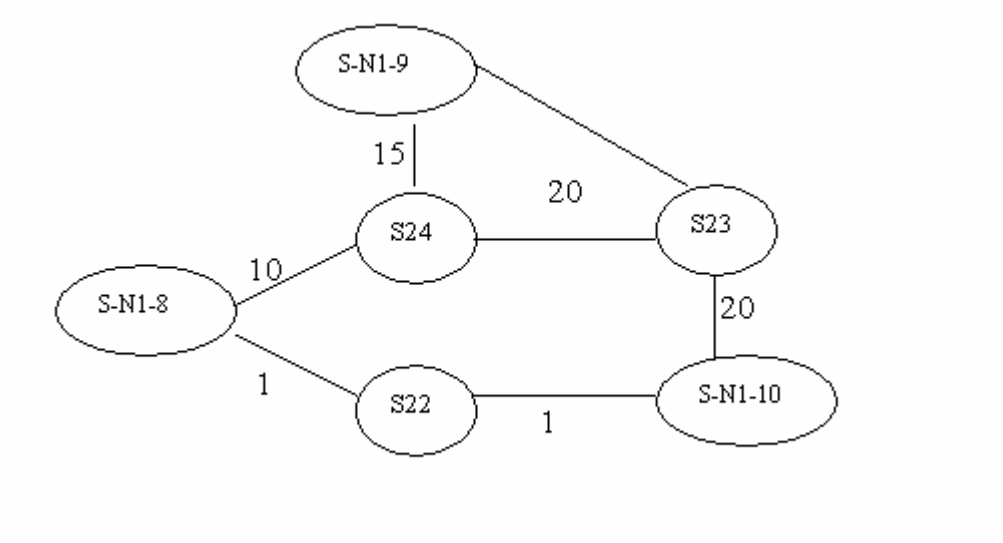

Figure 27: Topology of Network N3

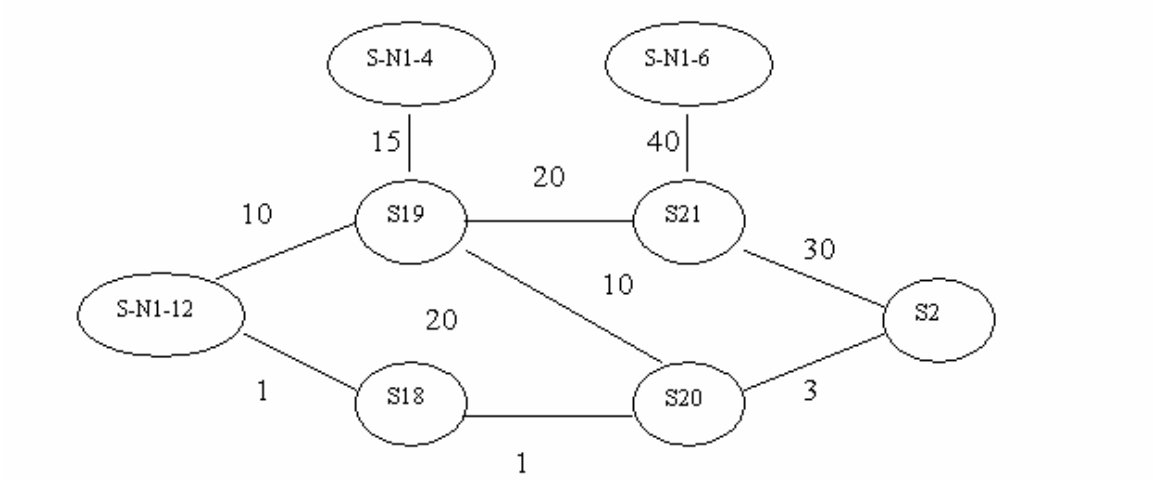

Figure 28: Topology of Network N4

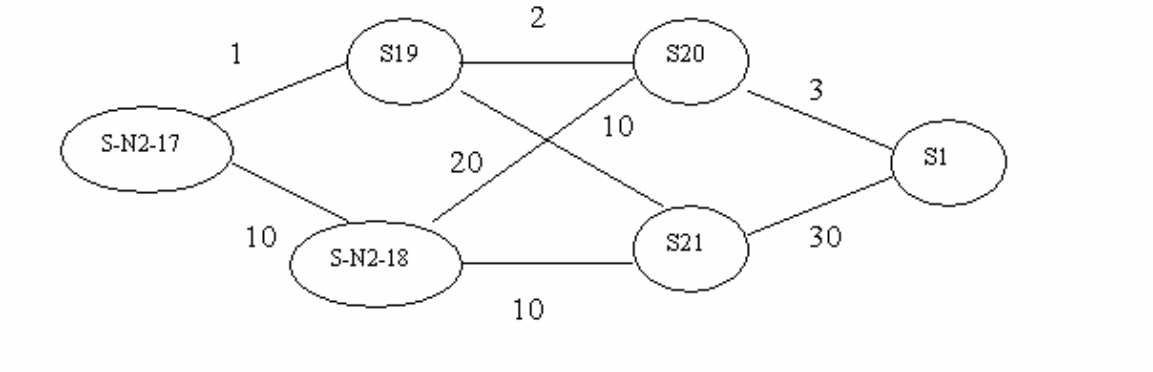

Figure 29: Topology of Network N5

The routing request is initially sent to network N5. Since the lowest common parent network is N1 for the two switches. The request is delegated to N2, and eventually N1. After analyzing the input addresses, which are Root/N1/N2/N5/S1 and Root/N1/N4/S2, network N1 understands that it must find a shortest path between network N2 and N4 first. It executes the routing algorithm and finds the following path: N2, S-N1-7, S-N1-8, N3, S-N1-10, S11, S-N1-12, N4.

The Web service operation is recursive due to the hierarchical nature of networks. When the routing alogrithm produces a path containing subnetworks, the parent network is responsible for invoking the routing operations of the subnetworks respectively to

establish the end-to-end connection. Since there are three subnetworks involved in the end-to-end connection, network N1 will send three routing requests to the subnetworks.

- It sends a routing request to N2 with the following addresses: Root/N1/N2/N5/S1 and Root/N1/N2/S-N1-7.
- It sends a routing request to N3 with the following addresses: Root/N1/N3/S-N1-8 and Root/N1/N3/S-N1-10.
- It sends a routing request to N4 with the following addresses: Root/N1/N4/S-N1-12 and Root/N1/N4/S2.

Network N2, N3, and N4 executes the routing algorithm to establish path segments. N2 will send a routing request to N5. When the routing process is finished, network N1 will send a respond message to the client indicating that an end-to-end connection has been established. Figure 30 shows the switches involved in the connection.

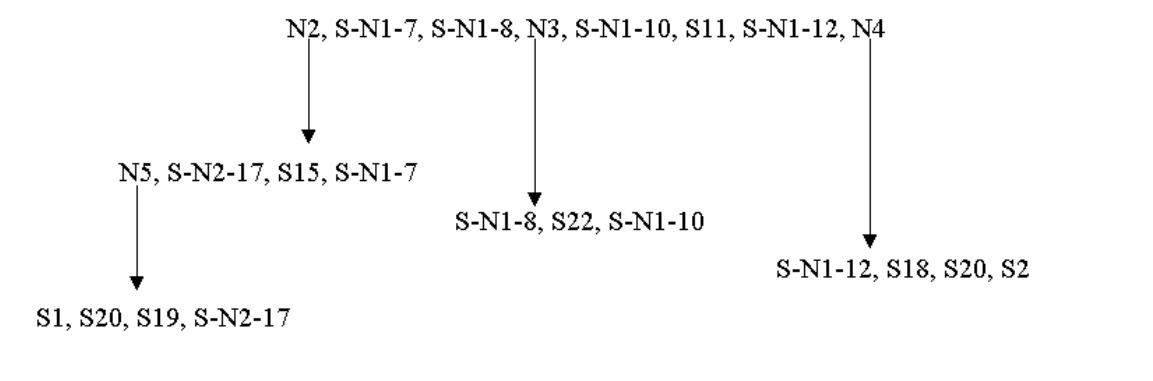

Figure 30: A List of Switches Involved in an End-to-End Connection Returned by Network N1

By observing the topological graph of each network, we can verify that the established path is the shortest path within each network.

Chapter 7 - Conclusion

The thesis describes the service-oriented architecture of the UCLPv2 system and the integration of new routing functionalities to automatically establish end-to-end connections. My major contributions are the following:

- I designed and implemented the intra-domain routing functionality, which is integrated into the UCLPv2 GUI. An end-user can establish end-to-end connections using resources belonging to his or her organization.

- I designed and implemented the inter-domain routing functionality. This is achieved through the introduction of a new network management Web service for each organization in UCLPv2. These Web services form a scalable network management hierarchy and collaborate to accomplish inter-domain routing. Consequently, end-users can establish end-to-end connections using resources belonging to different organizations. Furthermore, I built a client program into the UCLPv2 GUI so that end-users can make inter-domain routing requests. I also built an administrator client program to manage the network hierarchy.

The inter-domain routing framework presented in the thesis lays the foundation for future improvement.

- One improvement would be to add security and access control mechanism to the network management Web service, which should provide routing service only to authorized users or management entities.

- Another improvement can be made as part of the resource advertisement mechanism. Currently, a resource list file is modified manually and copied to a predefined Web service resource directory. This process can be improved by designing a tool in the GUI to export the file directly into the network management Web service.

References

[1] G. v. Bochmann, “Hierarchical inter-domain management for networks with condo-switches”, Proceedings IASTED International Conference on Communication Systems and Applications, Banff, Canada, Acta Press, pp. 190-196, July 2005.

[2] “What is UCLP”, http://www.uclp.ca, January 2007.

[3] S. Graham, D. Davis, S. Simeonoy, G. Daniels, P. Brittenham, Y. Nakamura, P. Fremantle, D. Koenig, C. Zentner, “Building Web services with Java”, Second Edition, Sams Publishing, 2005.

[4] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, “Web Service Architecture”, http://www.w3.org/TR/ws-arch/, February 2004.

[5] Tim Berners-Lee, “Uniform Resource Identifiers (URI): Generic Syntax”, RFC2396, http://www.ietf.org/rfc/rfc2396.txt/, August 1998.

[6] P. V. Biron, A. Malhotra, “XML Schema Part 2: Datatypes”, http://www.w3.org/TR/2000/CR-xmlschema-2-20001024/, October 2000.

[7] D. C. Fallside, P. Walmsley, “XML Schema Part 0: Primer”, http://www.w3.org/TR/xmlschema-0/, October 2004.

[8] E.Christensen, F. Curbera, G. Meredith, S. Weerawarana, “Web Services Description Language (WSDL) 1.1”, http://www.w3.org/TR/wsdl, March 2001

[9] Object Management Group, Inc. (OMG), “The Common Object Request Broker: Architecture and Specification”, http://www.omg.org/docs/formal/98-12-01.pdf, July 1995.

[10] N. Mitra, “SOAP Version 1.2 Part 0: Primer”, http://www.w3.org/TR/2003/REC-soap12-part0-20030624/, June 2003.

[11] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. F. Nielsen, “SOAP Version 1.2 Part 2: Adjuncts”, http://www.w3.org/TR/soap12-part2/#datamodel, June 2003.

[12] C. Peltz, “Web services orchestration”, http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestration.pdf, January 2003.

[13] “Introduction to the ActiveBPEL Engine”,http://www.activebpel.org/info/intro.html, January 2007.

[14] W. Stallings, “High speed networks and Internets: performance and quality of service”, Ch.15, Ch.16, Second Edition, Prentice Hall Publishing, October 2001.

[15] G. Malkin “Rip version 2”, RFC 2453, http://rfc.sunsite.dk/rfc/rfc2453.html, November 1998.

[16] E. W. Dijkstra, “A note on two problems in connexion with graphs”, Numerische Mathematik, pp. 269–271, June 1959.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, “Introduction to algorithms”, Second Edition, The MIT Press, pp. 595, 2001.

[18] M. T. Goodrich, R. Tamassia, “Data Structures and Algorithms in Java”, Second Edition, pp. 587, 2001.

[19] H. Zhang, M. Savoie, J. Wu, S. Campbell, G. v. Bochmann, B. St. Arnaud, “Service-oriented Layer 1 Virtual Private Network for Grid Applications”, Proceedings of the 2005 International Conference on Grid Computing and Applications (GCA'05), pp. 106-111, June 2005.

[20] H. Zhang, M. Savoie, J. Wu, S. Campbell, G. v. Bochmann, R. Liscano, M. T. Si, Q. Wang, B. Ho, S. Figuerola, G. Junyent, E. Grasa, J. Recio, A. López, Á. Sánchez, M. Lemay, “UCLPv2 detailed design document”, http://uclp.ca/twiki/bin/viewauth/UCLP/DetailedDesignDoc, April 2006.

[21] “UCLPv2 user manuals” http://uclp.ca/uclpv2/documents/help/uclpv2.0.2/, January 2007.

[22] A. Bolour, “Notes on the Eclipse Plug-in Architecture”, http://www.eclipse.org/articles/Article-Plug-in-architecture/plugin_architecture.html, July 2003

[23] M.Scarpino, S.Holder, S.Ng, L.Mihalkovic, “SWT/JFace in Action: GUI Design with Eclipse 3.0”, Manning Publications, January 2004.

[24] B.Sam-bodden, C.Judd, “Rich clients with the SWT and JFace”, APress, March 2004.