uOttawa

L'Université canadienne
Canada's university

An Implementation of Resource Advertising and Discovery for Optical Research Networks

Master Thesis By

Jun Jian

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 Electrical and Computer Engineering
School of Information Technology and Engineering
University Of Ottawa
Ottawa, Ontario, Canada

February, 22, 2010

©Jun Jian, Ottawa, Canada, 2010

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-69042-0
Our file Notre référence
ISBN: 978-0-494-69042-0

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Abstract

In optical research networks, the need to enable network users to dynamically provision and manage network resources inspired the development of the User Controlled Lightpath Provisioning System (UCLP). The UCLP version 2 (UCLPv2) is based on a service–oriented architecture in which network resources are abstracted as and managed via Web Services. It introduces a new concept, called Articulated Private Network, where end-users collect resources from several different domains to dynamically articulate and change their network topology.

In the current UCLPv2 system, the process of resource sharing is manual, usually error prone and inefficient. This thesis proposes, designs and implements a new functionality for resource advertising and discovery as an enhancement to the UCPv2 system. The new functionality describes the UCLPv2 resources based on the Resource Description Framework (RDF) and advertises them in directories .This allows automation of the resource advertising and discovery. By adopting RDF, the capability of sharing resources with other provisioning applications is achieved.

Acknowledgements

I would like to express my great appreciation to my supervisor, Dr. Gregor von Bochmann for his insightful feedback and motivational guidance. I am also grateful to Scott Campbell, Mathieu Lemay, Hanxi Zhang, who are the UCLPv2 development team members at Communication Research Center and Inocybe Inc, for their cooperation and support.

Table of Contents

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Figures vii

List of Abbreviations viii

Chapter 1 - Introduction 1

1.1. Background 1

1.2. Our contribution 3

1.3. Organization of the thesis 4

Chapter 2 – Service-Oriented Architecture and Web Services 6

2.1. Service Oriented Architecture 6

2.2. Web Services 8

2.2.1. XML 9

2.2.2. SOAP protocol 12

2.2.3. WSDL 14

2.4. Web Service Resource Framework 15

Chapter 3 - Service registries in SOA: an overview 18

3.1. UDDI 18

3.1.1. UDDI data model 18

3.1.2. UDDI programming API 20

3.1.2.1. Inquiry API 20

3.1.2.2. Publishing API 21

3.1.3. UDDI registry data replication 21

3.2. Jini lookup service 22

3.2.1. Discovery and join 23

3.2.2. Lookup service data Model 25

3.2.3. Lookup service interfaces 27

3.2.3.2. ServiceRegistration interface 30
3.2.4. Lookup process 31

3.3. Resource Description Framework 32
3.3.1. RDF data structure 33
3.3.2. RDF schemata 35
3.3.3. Querying RDF databases 36
3.3.4. Sesame 38

Chapter 4 - User-Controlled Lightpath Provisioning System 41

4.1. Example of use of a UCLPv2 system 44

4.2. UCLPv2 system architecture 48
4.2.1. Resource Management Service Layer 48
4.2.2. Resource virtualization service layer 49
4.2.2.1. Resource lists 51
4.2.3. High-level Services/User applications 54
4.2.3.1. RMC 54
4.2.3.2. Chronos 55

Chapter 5 - Designing resource advertising and discovery for UCLPv2 58

5.1. UCLPv2 resource sharing 60

5.2. System requirements 60

5.3. Design choices for resource advertising 62
5.3.1. UDDI 62
5.3.2. Jini lookup registry 63
5.3.3. RDF/Sesame 63
5.3.4. Inter-working consideration 64
5.3.5. Basic design choices for UCLP resource adverting and discovery 67

5.4. Design of resource advertising and discovery for UCLP 68
5.4.1. System architecture 68
5.4.2. Resource list transformation 69
5.4.2.1. RDF schema for UCLPv2 RL 72
5.4.3. Sesame public directory 76
5.4.4. Naming convention for RLs 77
5.4.5. Advertising, discovery and leasing policy 77
5.4.6. A typical use of resource list advertising and discovery 78
5.4.7. Functions for resource advertising, discovery and leasing 80

Chapter 6 - Detailed system design and Implementation 83
6.1. Sesame directory implementation 83
6.1.1. Implementing the directory manager 83
6.2. Client GUI program 84
6.3. XML to RDF 85
6.4. The RL downloading Web Service 88
6.5. Class level design of resource advertising and discovery functionalities 90
6.6. Consideration of access right of the directory 92
6.7. Implementation and test 93
6.7.1. Test of resource adverting, discovery and leasing 93
6.7.2. Performance test 96
Chapter 7 - Conclusion 100
References 102
Appendix A: RDF schema of UCLPv2 resource list 105
Appendix B: An example of a simple network described by NDL 111
Appendix C: WSDL for UCLPv2 RemoteDB-WS 113

List of Figures

Figure 1: Service participant relations ... 7
Figure 2: Web Service Architecture [1] ... 9
Figure 3: Build-in data type for XML schema [19] ... 11
Figure 4: SOAP message structure [2] ... 13
Figure 5: UDDI data model [5] ... 19
Figure 6: Sesame's Architecture [23] ... 39
Figure 7: CAnet4 physical network view in the UCLPv2 system (screen-shot) ... 44
Figure 8: Logical network view of CAnet4 ... 45
Figure 9: Resource List for NRC ... 46
Figure 10: NRC APN ... 47
Figure 11: UCLPv2 Service Oriented Architecture ... 49
Figure 12: UML class diagram of Lightpath and Interface resource ... 51
Figure 13: UML class diagram of Resource List data structure ... 52
Figure 14: Interactions between Chronos ... 57
Figure 15: UML class diagram of NDL topology schema ... 65
Figure 16: UML instance diagram of a NDF description of a simple network ... 66
Figure 17: System architecture ... 69
Figure 18: An UML diagram of RDF-RL data structure ... 70
Figure 19: UML sequence diagram for advertising a resource list ... 79
Figure 20: Sequence diagram for discovering a reserved resource list ... 80
Figure 21: System components of resource adverting and discovery ... 81
Figure 22: UML activity diagram for the procedures of resource advertising ... 85
Figure 23: Using an XSLT engine [27] ... 86
Figure 24: UML presentation of Web Service "RemoteDB_WS" ... 89
Figure 25: RemoteDBWSWrapper.java ... 90
Figure 26: UML class diagram of Java class for resource advertising and discovery classes ... 91
Figure 27: Resource lists adverting (screen-shot) ... 94
Figure 28: Resource discovery by reservation (screen-shot) ... 95
Figure 29: Uploading test result ... 96
Figure 30 Test result for lookup response time ... 98

List of Abbreviations

APN	Articulated Private Network
DRAC	Dynamic Resource Allocation Controller
DRAGON	Dynamic Resource Allocation via GMPLS Optical Networks
GLIF	the Global Lambda Integrated Facility
ITF-WS	Interface Web Service
LP-WS	Lightpath Web Service
NDL	Network Description Language
RDF	Resource Description Framework
RDF-RL	RDF-formatted Resource List
RDFS	RDF Schema
RL	Resource List
RMC	Resource Manager Console
RQL	RDF Query Language
SAIL	Sesame Storage And Interface Layer
SeRQL	Sesame RDF Query Language
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPARQL	Simple Protocol and RDF Query Language
UCLP	User-Controlled Lightpath Provisioning
UDDI	Universal Description Discover and Integration
W3C	World Wide Web Consortium
WS-A	Web Service Addressing
WSDL	Web Service Description Language
WS-Resource	Web Service Resource
WSRF	Web Service Resource Framework
XC-WS	Cross-Connection Web Service
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation

Chapter 1 - Introduction

1.1. Background

CAnet4 [31] is the Canadian nationwide research and education network owned and operated by CANARIE Inc. [31] Its main goal is to facilitate the development and use of Canada's advanced communications infrastructure. One of the characteristics of CAnet4 is that it may be considered to consist of many parallel networks running over the same physical substrate. This substrate is an aggregation of point-to-point 10 Gbps wavelengths leased from different carriers. The design and construction of the CAnet4 network is based on a hybrid network model. Optical hybrid networks consist of a routed IP part and a circuit-switched optical part (lightpaths). These hybrid networks lightpaths are provisioned through the network on demand. The lightpaths can be used to transfer large amounts of data, or to get a guaranteed fixed quality of service regarding bandwidth, delay, or jitter. Unfortunately, most lightpaths are provisioned manually, a process which may take days or weeks. It may take even longer if a lightpath goes through multiple network domains. Such lightpath provisioning requires clear communication between all the parties involved. It is acceptable for a provision to take weeks or months for a lightpath which may be in place for long periods of time. However, this is unacceptable for research projects such as sensor networks which need to download the data from multiple remote sensor nodes via one dedicated shared lightpath as well as Grid computing, E-science applications and high-definition video-on-demand streaming applications that require dedicated on-demand connections for intensive data transmission, as well as dynamic provisioning from time to time.

On the other hand, organizations such as universities, hospitals, schools and research institutes tend to buy or lease dark fibers and point to point links from different suppliers and would like to manage these resources as part of their own network domain. Furthermore, these organizations can sublease the resources to their users. 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 the ports on the switch, which is then called a condominium switch. They also may together lease or purchase fibers on which each organization acquires ownership of a subset of the wavelength channels. Condominium switches and shared fibers form the "condominium network" concept. Consequently, customer-owned networks are emerging where users control their network resources for the creation of connections and for sharing network resources.

To fulfill the requirements of dynamic provisioning and the management of customer-owned networks, the user-controlled lightpath provisioning project, UCLP [25], was put in place by CANARIE Inc. in 2002. In Phase 1, UCLPv1, the software allows a physical network to be partitioned into several independent management domains, and expose the network resources as software objects. These objects can be put under the control of different users so that they can create their end-to-end connections (lightpaths) without the intervention of the physical network operators. Users can dynamically provision lightpaths using their own resources to meet the requirements of user applications. The concept of user-controlled lightpath provisioning has been verified to be successful in Phase 1. In order to fulfill the requirement of customer-owned network management, UCLPv2 was put in place in 2005. 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. The UCLPv2 system uses the concept of Articulated Private Network (APN) [33] to allow users to define complex multi-domain network topologies by binding together network resources from several domains. An APN topology can be setup for each specific application. Furthermore, by abstracting network resources into Web Service resources and exposing its functions via Web Services, the UCLPv2 system enables third-party applications to build their network services based on the UCLPv2 Web Services and resources.

UCLPv2 allows organizations to share their free resources by exporting a resource list (RL) to other organizations. The exported resource lists, which contain resources, can be

imported by other organizations to create new RLs or APNs. The end-users may utilize these imported resources to create end-to-end connections. Resource sharing is greatly beneficial for efficient resource utilization and the creation of inter-domain end-to-end connections. To create inter-domain connections, one organization needs to acquire the desired resources from other organizations by importing them in resource lists. All the administrators of the involved organizations have to negotiate and coordinate the steps. The resource lists are usually transferred via email or FTP applications after the agreement has been achieved through some informal conversations. This process of resource sharing is manual, usually error prone and inefficient. In order to automate this process, a suitable resource advertising mechanism is needed.

1.2. Our contribution

The Global Lambda Integrated Facility (GLIF) [10] is an international virtual organization of research networks, research consortia and institutions whose aim is to build a worldwide networking facility for scientific research. GLIF consists of a collection of optical exchange points, GOLEs (GLIF Open Lightpath Exchanges) and links between them. A GOLE node is an optical switch that works like a condominium switch in which the resources are shared by GLIF organizations. A global network is formed through lightpath connections to GOLEs. The global network currently consists of over a dozen GOLEs spread over North America, Europe, East Asia, and Australia, with numerous links across and between the continents [17]. CANet4 is a member of GLIF. The UCLP system has been used for lightpath provisioning for GLIF research projects.

In the context of GLIF, Lightpaths always need to be provisioned among several domains. Several parties realized that in order for lightpaths to scale, the provisioning process must be (partly) automated. Currently there are a few applications available to do provisioning and brokering for GLIF organizations, such as UCLP, DRAGON (dynamic resource allocation via GMPLS optical networks) [15], and DRAC (dynamic resource allocation controller) [14]. But none of them can inter-work with the others for provisioning. One important aspect is

that applications, such as UCLP, DRAC, at least should know the path information across multiple network domains for automated lightpath provisioning and brokering. To achieve this, the Network Description Language (NDL) [13] is proposed by GLIF to describe network topologies and resources using RDF. Different provisioning applications are able to understand each other by using the same language to describe network topologies and resources. Consequently, inter-working is achieved. As a network engineer for CANet4, I have participated in the development of NDL to define the NDL schemas, where vocabularies have been defined for describing network topologies.

To address the resource advertising and discovery problem within the context of UCLPv2, this thesis focuses on a solution of resource adverting and discovery based on the Resource Description Framework (RDF) and the Sesame directory [34]. The solution uses RDF to describe the UCLPv2 resource lists and the contained resources by transforming the XML resource lists into RDF format (RDF-RLs). The RDF-RLs are advertised in the Sesame directory either to the public or to a specific organization. The latter is called resource reservation. Administrators of UCLPv2 organizations may discover resource lists that are advertised publicly or reserved for their organizations. This solution is compatible with the NDL approach since the NDL network topology schema was adopted for the RDF descriptions of UCLPv2 resources and the resource lists. The implementation is built using the Sesame APIs and Web services technology in accordance with the service-oriented architecture of UCLPv2. Furthermore, a client GUI program was developed and implemented to provide the UCLPv2 users a graphical interface for resource adverting and discovery.

1.3. Organization of the thesis

The goal of this thesis is to design a resource advertising and discovery mechanism based on the current UCLPv2 system that facilitates organizations to share resources and automate the resource acquiring process. To foresee the growth of the number of organizations participating in resource sharing/advertising, the advertising solution must be

scalable. Furthermore, the solution must also support interoperability with other applications, such as DRAC, DRAGON, indirectly via Chronos [32].

The rest of this thesis is organized as follows. Chapter 2 provides background information on service-oriented architecture and Web services technology. Chapter 3 reviews some common service registries proposed for SOA systems including the RDF/Sesame registry. Chapter 4 presents the system architecture and the resource list data model of the UCLPv2 software. In addition, the chapter briefly introduces the UCLPv2 GUI which is built on the Eclipse Rich Client Platform by presenting an example of using UCLPv2 on CANet4. Chapter 5 explains the manual process of resource sharing in current UCLPv2 systems, describes the system requirements for resource advertising and discovery as well as some important design decisions. Furthermore, it explains how to derive a RDF data model from UCLPv2's XML resource lists and defines functionalities for resource advertising and discovery. A simple scenario of inter-working between UCLPv2 and other applications is presented in this chapter too. Chapter 6 focuses on the implementation of the mechanism of resource advertising and discovery. Finally, chapter 7 concludes the thesis and presents some ideas for future improvement.

Chapter 2 – Service-Oriented Architecture and Web Services

2.1. Service Oriented Architecture

Service Orient Architecture, SOA, is an architecture for distributed system modeling in which the software components are modeled as services conforming to well-defined interfaces. An interface defines what functionalities a service provides and the protocol for accessing these functions. Applications knowing the interface of a service can freely invoke the service without knowing the underlying detailed implementation of the service. This is achieved by interfaces abstracting away underlying complexity of services. The emphasis is on the design of interfaces rather than the implementation of software components. A well-designed interface allows new versions of services to be introduced without breaking existing services, which greatly reduces dependency among services which in turn makes SOA system a loosely coupled system.

One advantage of a loosely coupled system lies in the ability of the system to adapt to changes. Giving the fact that the software development life cycle is iterative, and new functional requirements are inevitable, designing a system to be loosely coupled is important.

A service exposes its functions via an interface. Other applications invoke the service by knowing the service's interface. This promotes the reusability of SOA systems. The same service can be invoked by different applications while providing the same external behavior in accordance with its interface definition. The reusability greatly reduces codes of programming which complies with the object-oriented design paradigm which is "code once and use everywhere". Since the relationship between services is solely based on the interface definitions, services do not need to know how and where the companion services are implemented. In other words, services do not rely upon a uniform system or a

homogeneous runtime technology. With a SOA, services can be composed to represent a workflow – e.g., a whole business process or a complex science experiment.

In a loosely coupled SOA system, Services are independent of each other. Services locate other services via a directory service. A directory is a place where entities (service providers and service requesters) can register and search for services. A service directory is also a service with a well-known interface. The service directory works as a broker between service providers and service requesters or consumers. The directory service implements a look-up mechanism where consumers can go to find a service based on some criteria. The relationship between the different parties is depicted in Figure 1

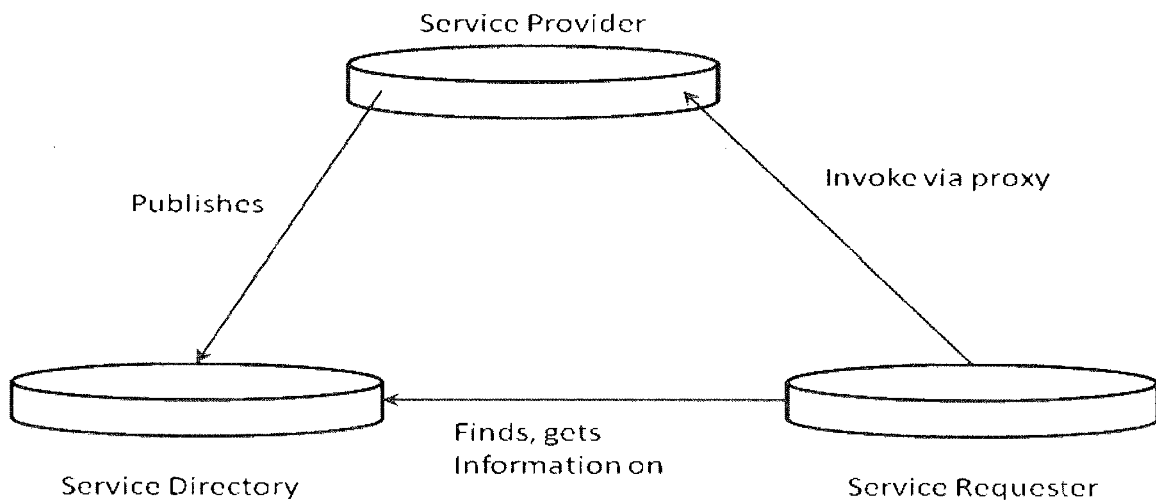

Figure 1: Service participant relations

Service providers register with the directory service whereas consumers query the directory service to find service providers. Most directory services typically organize services based on some criteria and categorize them.

Embedding a directory service within SOA accomplishes the following:

- Scalability of services; services can be added incrementally.
- Decouples consumers from providers.
- Allows for hot updates of services.
- Provides a look-up service for consumers.
- Allows consumers to choose between providers at runtime rather than hard-coding a single provider.

2.2. Web Services

Web services play a major role in SOA. According to World Wide Web Consortium (W3C) definition, a Web service is a software system designed to support interoperable machine-to-machine interaction over a network using a specific protocol, called Simple Object Access Protocol, SOAP [2]. It has an interface described in a machine-processable document called Web Service Description Language file, WSDL [3]. A WSDL is an XML document which describes the information of Web services and specifies how to access the service. Other systems interact with the Web service in a manner prescribed by its WSDL using SOAP message, typically conveyed using HTTP and XML serialization in conjunction with other Web-related standards.

The main goal of the Web Service standard is to support interoperability between different applications. As described in the Web Service definition from W3C, Web Services comply with the architecture of SOA. Services are abstract resources with functionalities described by interfaces. Web Services can be implemented on different platforms and using different programming languages adhering to the relevant set of Web Service's specifications. Figure 2 illustrates an abstract view of various specifications that define the Web Services architecture stack. At the bottom of the stack is the communication layer. Transport protocols such as HTTP, SMTP, FTP can be used to transfer Web service messages, which are encapsulated in SOAP envelops. The interface description layer uses the WSDL standard to specify service interface definitions. The process layer includes several specifications for Web services discovery, orchestration, etc. XML, SOAP, WSDL, UDDI are the fundamental of Web Services technology. The following sections discuss these technologies in more detail.

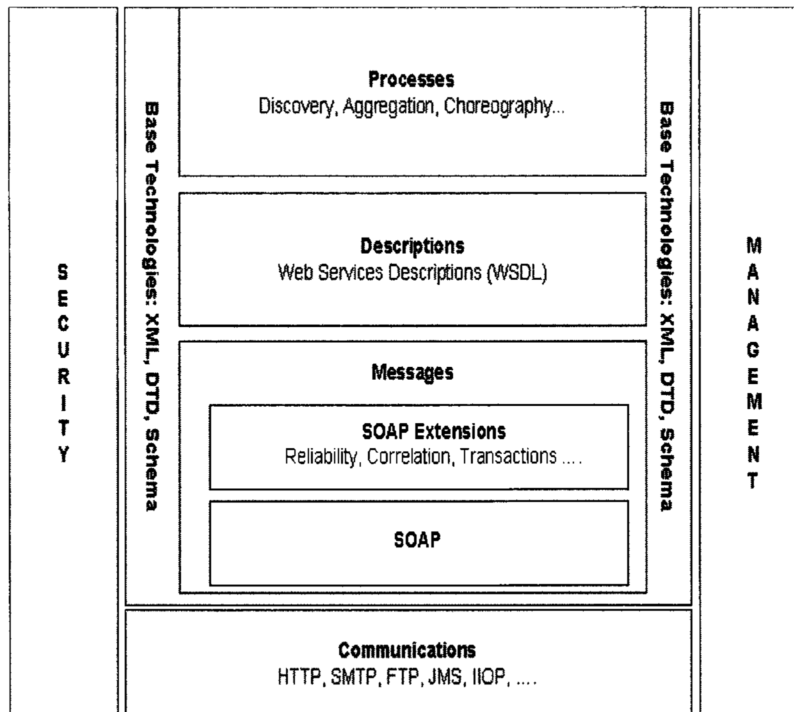

Figure 2: Web Service Architecture [1]

2.2.1. XML

XML stands for Extensible Markup Language which is used extensively within Web services as a standard, flexible, extensible data format. W3C recommends XML as an industrial standard with a set of definitions for structuring, storing and transporting information. XML is neither platform- nor vendor-specific.

An XML documents is a tree structure of nested elements each of which can have one or more attributes. Each element has a starting and ending tag, marked by angled brackets with content in between. There can only be one root element. The tags are not predefined. The content can contain other elements, or can consist entirely of other elements, or might be empty. Attributes are named values which are given in the start tag, with the values surrounded by single or double quote characters.

In order to avoid clashes between names from different markup elements/attributes, XML uses a mechanism of namespaces to provide uniquely named elements and attributes in an XML document. Elements with the same name, but different namespace, can be used in the same document. A fully qualified element name consists of a namespace identified by an URI (Uniform Resource Identifier) and a local name. It is used to uniquely identify resources that can be web-accessible.

A well-formed XML document is a document that conforms to XML syntax. However, this does not necessarily mean that a well-formed XML does not contain errors. The W3C XML specification states that a program should stop processing an XML document if it finds an error. The reason is that XML software should be small, fast, and compatible [19]. It is important to validate an XML document to make sure it conforms 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.

An XML Schema describes the structure of XML documents as a linearization of tree structure of nested elements and defines each element which can have one or more attributes. It also provides a data typing system to define vocabularies embedded in XML document.

To be specific, an XML Schema [20]:

- Defines elements that can appear in a document.
- Defines attributes that are associated with an element type.
- Defines the child relationship between elements and the order of child elements and their number.
- Defines whether an element may be empty or can include text
- Defines data types for elements and attributes
- Defines default and fixed values for elements and attributes

W3C has defined a hierarchy of data types to be used in XML schemas as shown in Figure 3.

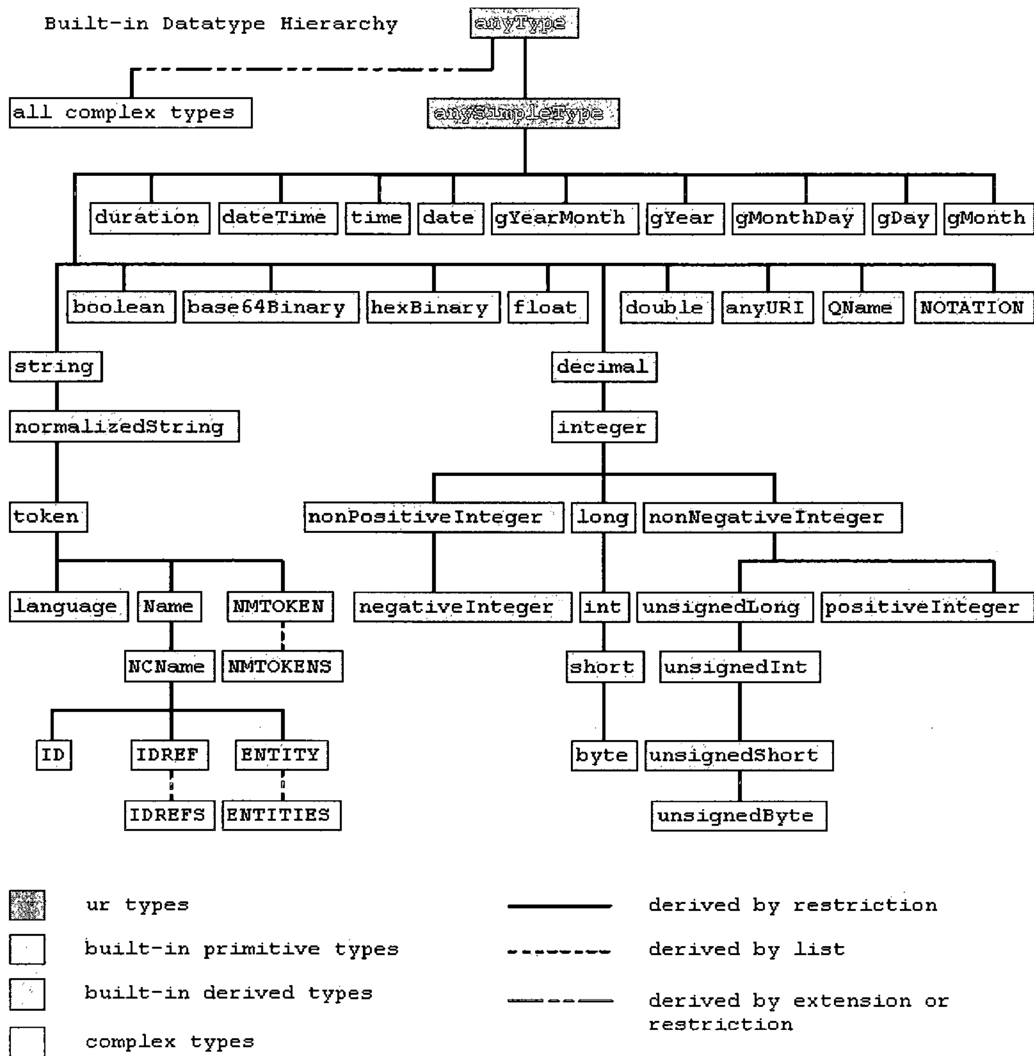

Figure 3: Build-in data type for XML schema [19]

The root of this hierarchy is anyType, which is extended into two groups: any simple type and all complex types. Complex types are all user-defined types. Simple types include string, integer and date as build-in types. One main difference is that complex data types may contain nested XML elements and carry out attributes; simple types cannot have XML elements and cannot carry attributes. Furthermore, one can specify the sequential 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 on which the new type is based [2].

The following is an example of defining a simple data type called countryType by restriction:

```
<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>
```

In an XML schema file, an element could be defined to have the name "country" which has a type "countryType" as showed in the following:

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

The countryType is defined 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>
```

2.2.2. SOAP protocol

The Simple Object Access Protocol (SOAP) is an XML-based communication protocol which provides a standard, extensible, composable framework for packaging and exchanging XML messages between distributed applications. It is platform and language independent. SOAP is also independent of the underlying transport protocol, but is most commonly carried over HTTP. The SOAP message structure is depicted in Figure 4 [2].

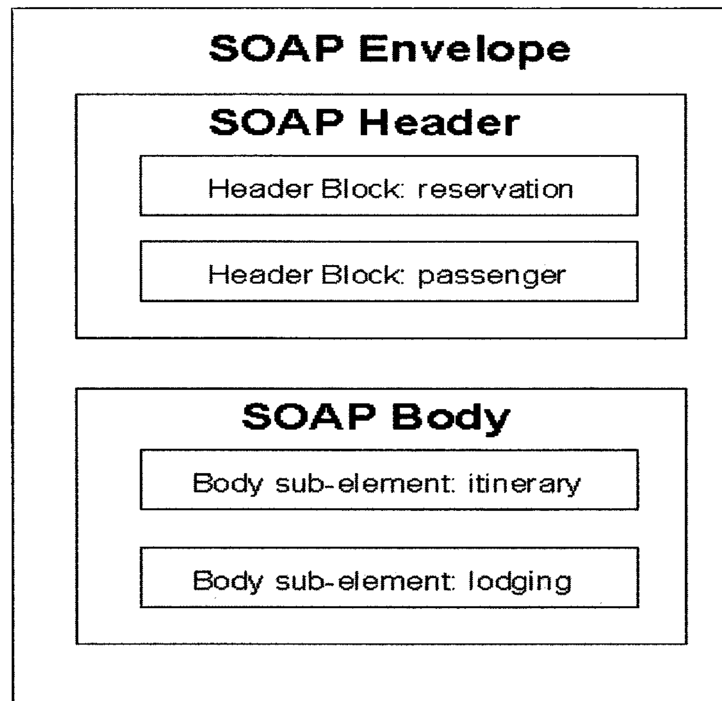

Figure 4: SOAP message structure [2]

A SOAP message consists of an envelope, a header and body elements. The header and body are encapsulated in the envelope. The header is optional, whereas the body is mandatory. A SOAP message may contain more than one header, but it can only have one body element. A SOAP header is an extension mechanism that provides a way to pass information that is not application payload. Such information includes, for example, directives or contextual information related to the way the message should be processed. This allows a SOAP message to be extended in an application-specific manner. SOAP headers can be inspected, deleted or forwarded by SOAP nodes encountered along a SOAP message path without affecting the content of the body. A SOAP body consists of the data content to be exchanged between applications. In case faults arise in the processing of a message, SOAP protocol embeds the fault messages in the SOAP body.

The purpose of the SOAP messaging and XML encoding is to provide a mapping of non-XML application-level data structures to XML representations for transmission. For example, Java objects can be transformed and serialized into XML elements and sent down the wire, and on the receiving side, the XML data will be un-serialized and transformed into Java objects.

The SOAP specification also specifies a protocol binding framework. A binding implies a set of rules for transferring a SOAP message within or on top of a specific transport protocol (underlying protocol). One of the most common binding is SOAP over HTTP, which takes advantage of the request-response mechanism of HTTP to achieve two-way communication.

2.2.3. WSDL

WSDL (Web Service Description Language) defines an XML format for describing Web Service interfaces. In a summary, a **Service** exposes groups of **operations**. Each group of operations is called **portType**. To invoke an operation, the service requester sends an input **message** and may get back an output message. **Types** define the data types that are relevant for the exchanged messages. The actual protocol used to invoke an operation and the input and output messages are specified in a **binding**. The service itself is exposed to outside via one or more ports. Each **port** specifies a network address where it is accessible and the binding used with this port. A service may have multiple ports each with different bindings. Since WSDL is based on XML, it is platform-neutral and language-independent.

A WSDL document is an XML document conforming to the WSDL schema definition. It uses the following major elements in the definition of Web Services [3]:

- **Types** – a container for data type definitions following the XML schema format.
- **Message** –an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- **Port Type** – which is a set of abstract operations. The messages associated to an operation are determined by the style of the operation. There are four operation styles.
    - **One-way** . The operation receives a message.
    - **Request-response**. The operation receives a message, and sends a correlated message.
    - **Solicit-response**. The operation sends a message, and receives a correlated message.
    - **Notification**. The operation sends a message.
- **Binding** – a transmit protocol and data format specification for a particular port type.
- **Port** – specifies an address for a binding, thus defining a single communication endpoint.
- **Service** – which is used to aggregate a set of related ports

There are two optional elements in WSDL documents as additions to the major elements listed above:

- **Documentation** - a container for human readable document.
- **Import** - reference to the definitions of Web service components that belong to other namespaces. It allows one WSDL document to import other WSDL documents and use any elements already defined in those namespaces.

In a summary the structure of a WSDL document looks like this:

```
<definitions>

<types>
 definition of types
</types>

<message>
  definition of a message
</message>

<portType>
        <operation>......<operation>
        <operation>......<operation>
</portType>

<binding>
  definition of a binding
</binding>

<service> definition of a service...
        <port> ...</port>
</service>

</definitions>
```

2.4. Web Service Resource Framework

The Web Service architecture defines a service-orient distributed computing model in which services interact by exchanging messages in XML format. Very often Web Services need to manipulate state, i.e. data values that persist across, and evolve because of Web Services interactions. For the purpose of promoting interoperability among Web Services and their interactions with stateful resources in a standard way, Web Service Resource Framework (WSRF) specifications have been defined and finalized by the OASIS working

committee to define a generic and open framework for modeling and accessing stateful resources using Web Services.

WSRF models stateful resources through the WS-Resource concept. A WS-Resource is composed of a Web Service and a stateful resource. Stateful resources have a well-defined life cycle and can be described by an XML document. Such an XML document is named "resource properties document". It consists of elements representing resource properties. A Web Service exposes its resource properties by associating resource properties document with the Web service WSDL portType definition [4] in its WSDL document.

A WS-Resource is identified through the use of a WS-Addressing endpoint reference (EPR) which is composed of the URL of the Web Service and a key that uniquely indentified the resource within the Web Service. By specifying the EPR of a WS-Resource, Web services can interact to retrieve, change and delete resources and their properties through message exchanges. A WS-Resource can be created via a Web service referred to as resource factory. The EPR is returned as the result of the WS-Resource creation by the resource factory.

WS-Resources have a well-defined lifecycle. WSRF specifications define mechanisms for WS-Resource destruction, including message exchanges that allow a service requester to destroy a resource, either immediately or by using a time scheduled resource termination mechanism [4]. WSRF focuses on the definition and modeling WS-Resources. The detailed implementation of such resources varies according the technology used. A specific resource state may be stored in memory, in a file system, in a database, or in some XML repository. It may be coded as an XML document or any other implementation-dependent format.

The UCLPv2 system (see Chapter 4) utilizes the WSRF concept to abstract physical network resources into logical resources such as lightpath resources. Lightpath resources are created via a lightpath resource factory. A lightpath has two endpoints which represent the two ports linked by the lightpath on the two network elements. An endpoint is abstracted as a WS-resource named endpoint resource and created by the lightpath resource factory. An

EPR is generated in the process of creating an endpoint resource. A lightpath resource is formed by associating its two endpoint resources EPR together with its properties such as the bandwidth, the direction and the type. A unique key is used to identify the lighpath resource.

Chapter 3 - Service registries in SOA: an overview

Service registries play an important role in SOA systems. A service registry provides a common directory service and acts as a broker between service providers and consumers. By embedding service registries in SOA systems, services can be added incrementally which provides for system evolution. Knowing the service registry, service consumers do not need to hard-code a single service provider, instead, service consumers choose a service provider through the use of the look-up service provided by a service registry. Service consumers are decoupled from service providers which allow services to be hot updated without changing the system architecture. There are many types of registries that have been proposed for SOA systems. In this chapter, UDDI [5], Jini [7] and RDF [34] registry will be reviewed.

3.1. UDDI

UDDI stands for Universal Description, Discovery and Integration. It is the standard for web service discovery and is, with SOAP and WSDL, one of the fundamental standards for Web Service. UDDI specifications describe a registry of Web Services and a set of programmatic interfaces to support the description and discovery of businesses, organizations, and other Web services providers as well as the Web services they make available and the technical interfaces which are used to access and manage these services. In fact, UDDI itself is a set of Web Services and is based upon several well-established industry standards such as HTTP. UDDI defines a data model and a set of programming APIs which are introduced in the following sessions.

3.1.1. UDDI data model

The UDDI data model has five core data types which make up the information provided within the UDDI service description. Figure 5 shows the data model defined by UDDI 2.0

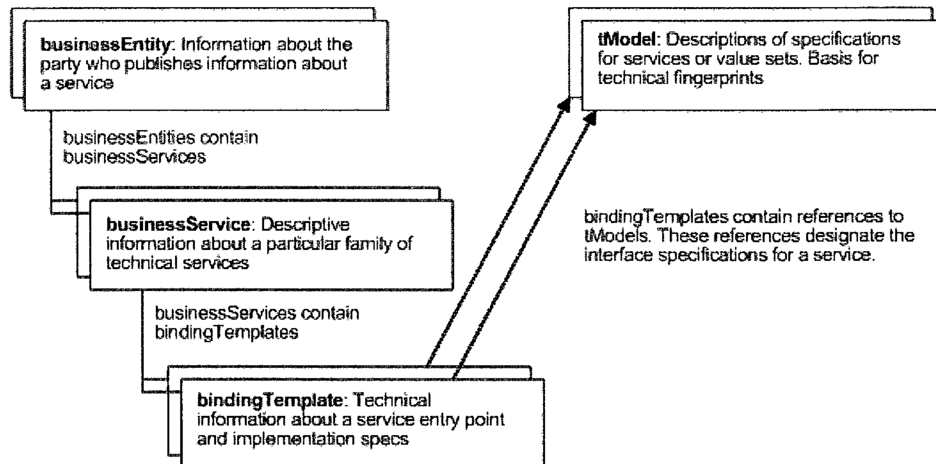

Figure 5: UDDI data model [5]

Each data structure is identified by a Universal Unified ID, UUID. A UDDI registry assigns these unique identifiers when information is first saved, and these identifiers can be used later as keys to access the specific data instances on demand.

The core data types are related by containment as shown in Figure 5. The businessEntity structure contains one or more unique businessService structures. Similarly, individual businessService structures contain specific instances of bindingTemplate data, which in turn contains information that includes pointers to specific instances of tModel structures.

An item in an UDDI registry consists of the following parts:

- **businessEntity**: Describes a business or an organization that provides Web Services. The descriptive information contains the name of the business, a point of contact such as the address, telephone number, email and fax number, the URL of the company web site and so on.
- **businessService**: Describes a collection of related Web Services offered by an organization described by the businessEntity.
- **bindingTemplate**: Describes the technical information necessary to use a Web Service. Since UDDI's main purpose is to enable description and discovery of Web Service information, it is the bindingTemplate that provides the most interesting technical data. Each bindingTemplate structure has a single logical businessService parent, which in turn has a single logical businessEntity parent.
- **tModel**: Describes a "technical Model" for Web services in the UDDI registry, including various other attributes such as taxonomy, transports, digital signatures [5].

3.1.2. UDDI programming API

UDDI specifies application programming interfaces (APIs) for service providers and service consumers to interact with the registry [6]. The Inquiry API is for consumers to find a service and the Publishing API is for service providers to register a service, it allows storing or changing data in the registry.

3.1.2.1. Inquiry API

Inquiry API supports three query patterns. The browse pattern allows service consumers to perform a search based on some broad information, find general result sets and then select more specific information for drill-down. This is done by way of *find_xx* (find_binding, find_business, find_ltservice, find_tModel) API function calls. For example, one can find whether a particular business has any information registered. Or one can find all the services that comply with a particular technical fingerprint (tModel signature) from a particular business. The drill-down pattern will let the service consumers get the full registered details for a specific data instance by using the key of that specific data item. This is done by *get_xx (get_bindingDetail, get_businessDetail, get_businessDetailExt, get_serviceDetail, etc.)* calls. The purpose of the invocation pattern is to find the new access point information in a bindingTemplate in case the associated remote service is moved and is unknown to the consumer. Generally, the bindingTemplate data obtained from the UDDI registry contains the details about an instance of a given interface type, including the location where a program can call the Web Service. This information is cached in the calling application. The calling application uses this information to construct a proxy then to interact with the remote service at the registered address via the proxy. The call will fail when a service is moved without the knowledge on the part of the callers. When a call fails using the cached information previously obtained from a UDDI registry, the proper behavior is to query the UDDI registry for a fresh bindingTemplate. The proper call is *get_bindingDetail* passing the original bindingKey value. If the returned data is different from the cached one, the service invocation will automatically retry the invocation using

the fresh information. If the retry is successful, the new information will replace the cached information. By using this pattern with Web services in UDDI registries, a service provider can strengthen the reliability of the services they provide.

3.1.2.2. Publishing API

The publisher API provides operations to create/update or delete core data type instances of the businessEntity, businessService, bindingTemplate and tModel. Detailed information about the Publisher API is given in [6].

A general use case for registering a service in a UDDI registry involves the following steps:

1. Choose a UDDI operator with whom to work.
2. Build or obtain UDDI client software. The UDDI operator should provide software to the client.
3. Obtain an authentication token (password) from the operator for security and authentication purpose. Authentication tokens are required for each publisher interface operation call.
4. Register the information about the business such as businessEntity, businessService, bindingTemplate, tModel and etc.
5. Testing
6. Update the information as necessary to reflect any changes.

3.1.3. UDDI registry data replication

The UDDI Business Registry (UBR), also known as the Public Cloud, is a conceptually single system built from multiple UDDI nodes that have their data synchronized through replication. The cloud services provide a logically centralized, but physically distributed, directory. This means that data submitted to one root node will automatically be replicated across all the other root nodes.

A single UDDI node that participates in the UBR, or the Public Cloud, is called an operator node. An operator node may consist of multiple machines to provide UDDI services. Multiple operator nodes form the UBR. The data custody concept is used in the process of

replication. Data custody means that only the operator node can modify the data that was created in its node and the node is responsible for initializing the replication process when the data is changed. Each data change will cause the operate node's USN (Update Sequence Number) [6] to be increased and a change record is generated. The change record consists of the UUID of the operator node, the updated USN resulting from the data change and the payload of the changed data. The USN is not only used to indicate the change of the data, it is also used to form a high water mark vector. The high water mark vector contains the USN of the most recent change record that has been successfully processed from each operator. This vector has one entry for each node in the replication graph. When a data change happens, the originating node sends it highWaterMark vector to its adjacent nodes to inform them about the data change. When an operator node receives the highWaterMark vector, it then compare the USN in the received vector with the one it keeps. If the operator node finds any node's USN changed, a request of get_changeRecords will be sent to its upstream node to pull the changed data. Then the highWaterMark vector is sent to this node's downstream node. The process continues till all nodes get the changes. The data replication path is determined by the communication graph that defined in the replication configuration file for the cloud.

3.2. Jini lookup service

A Jini system is a distributed system based on the idea of federating network services and clients within a SOA system implemented in Java. Jini systems have a set of components that provides an infrastructure for federating services [7]. A programming model has been defined based on the Jini architecture to support and encourage the production of reliable distributed services. The most important concept within the Jini architecture is that of a service. A *service* can be anything that sits on the network and is ready to perform a useful function. Hardware devices, software, communication channels -- even human users themselves -- can be services. Servers can be collected together to perform a particular task. Services are found and resolved by a Jini lookup service which acts as a service registry. The *Jini lookup service* is the central bootstrapping mechanism for the system and provides

the major point of contact between the system and the users of the system [7]. A Jini lookup service itself is a service. It could also register itself into another lookup service for the purpose of forming a hierarchical lookup mechanism. In the following, we call a Jini lookup service simply as lookup service in the rest of this session.

3.2.1. Discovery and join

A service is added into a lookup service by using the discovery and join functions. First, the service uses discovery protocols to locate a proper lookup service, and then register using the join protocol. At the end of the discovery process, a proxy of the discovered lookup service will be transferred to the discovering entity. The proxy is an object named service registrar which implements the net.jini.core.lookup.ServiceRegistrar interface. The service registrar interface defines methods to register services, find services, and receive event notification in case that modification happened in the lookup service. The interface also defines methods to incrementally explore the collection of items in the lookup service registry according to certain criteria. Several discovery protocols are foreseen, such as the multicast request protocol, the multicast announcement protocol and the unicast discovery protocol. The discovery process of different discovery protocols is described in the following.

When a service is initially started up, it uses the multicast request protocol for sending out multicast packets to a well-known port. The packet contains information such as an IP address and a port number by which the service can be contacted. Lookup services monitor the well-known port for these multicast packets. The packet contains information that enables the lookup service to determine whether or not it should contact the sender of the packet. If so, it contacts the sender directly by making a TCP connection to the IP address and port number extracted from the packet. Over this connection the lookup service sends its proxy--an object, called a *service registrar,* to the originator of the packet using the join protocol which is Java RMI. The purpose of the service registrar proxy is to facilitate further communication with the lookup service. By invoking methods on this object, the sender of the announcement packet can perform join and lookup on the lookup service [7].

The multicast announcement protocol is used by lookup services for announcing their presence in the Jini system. In other words, lookup services are required to announce live message to all the participants in the system throughout their lifetime. The announcement is done by sending, at regular intervals, a packet to a well-known port using the multicast UDP protocol. The packet consists information about the ID of the lookup service, the group value, the IP address and the port that the lookup service can be contacted via the unicast protocol. When a multicast announcement packet is received, the discovering entity determines whether or not to attempt unicast discovery to the announcing lookup service as follows: if the announcement's lookup service ID is contained in the set of service IDs from which the discovering entity has already heard, then the discovering entity must not perform unicast discovery. If the member groups listed in the announcement do not intersect with the discovering entity's groups of interest, then the discovering entity must not perform unicast discovery [8]. Otherwise, the discovering entity uses the unicast discovery protocol to get the lookup service's proxy—the object of service registrar.

Unicast discovery protocol is used when the lookup service is known to the services, the discovering entity. This protocol is fundamentally a simple request-response protocol. In the process of unicast discovering, a discovering entity opens a TCP connection to the lookup service's unicast IP address and port. Then the discovering entity sends a unicast discovery request over the connection. If the lookup service determines that the request is acceptable, it responds by transmitting its proxy—the object of service registrar over the connection to the discovering entity. The unicast discovery protocol is also employed as the last step of multicast discovery, after a connection to the lookup service's unicast IP address and port which has been established through either the multicast announcement or multicast request protocol [8]. The detailed specification of discovery protocols can be found in [8].

After the service discovers the lookup service and obtains the service registrar object, it is ready to do a join—to become the part of the federation of services that are registered in

the lookup service. It is where the join protocol is used. The join protocol makes use of the discovery protocols to define a standard sequence of steps that the service should perform when starting up and registering themselves with a lookup service [8]. In practice, to do a join, the service invokes the ***register ( )*** method on the service registrar object, passing an object called a service item as a parameter. The service item object is a bundle of objects that describe the service. The ***register ()*** method sends a copy of the service item up to the lookup service, where the service item is stored. Once this has completed, the service provider has finished the join process: its service has become registered in the lookup service.

3.2.2. Lookup service data Model

Lookup services maintain a flat collection of services items [9]. Each service item represents an instance of a service. Service items are instances of the *ServiceItem* class. The following is the sample of *ServiceItem* class definition.

```
public class ServiceItem implements Serializable {
    public ServiceItem(ServiceID serviceID,
                       Object service,
                       Entry[] attributeSets) {...}
    public ServiceID serviceID;
    public Object service;
    public Entry[] attributeSets;
}
```

Three public properties are defined in the *ServiceItem* class.

- ***service***: is a proxy of the registered service.
- ***serviceid***: is an instance of *ServiceID* class. Every service is assigned a universally unique identifier (UUID) called service ID. The service ID is either created at manufacture-deployment-time, or generated dynamically by lookup services at registration time. A service ID is a 128-bit value.
- ***attributeSets***: is an instance of an array of *Entry*. Each *"Entry"* represents an attribute set. Each individual attribute set is represented as an instance of a Java class with each attribute being a public field of the class that implements the interface Entry (*net.jini.core.entry.Entry*) [9]. The *attributeSets* array may contain multiple instances of the same class with different attribute values, as well as multiple instances of different classes. For example, an item might have multiple

instances of a *Name* class, each giving the common name of the service in a different language, plus an instance of a *Location* class, an *Owner* class, and various service-specific classes [9]. The main purpose of the *attributeSets* property is to provide additional, more detailed description of the service. The *attributeSets* is also used in the template matching lookup process.

Each service registration is associated with a lease. A lease defines the life time of the registration in the lookup service. Services are responsible for renewing their lease duration. If a service item's lease is expired and is not renewed before the expiration, the lookup service will cancel the item's lease and the service is deregistered. If a service wishes to end its registration with the lookup service, it will cancel the lease, and the lookup service will then drop the registration of that service. In case the service were to crash, or become unavailable because of network failure, it would lose the ability to cancel its registration. As a result of the failure, the lease for the registration will expire. At that point, the lookup service will drop the registration, reflecting the fact that the service is no longer available. This behavior is specified by the lookup service, so if the service becomes available after such a lapse, it knows that it needs to re-register (rather than try to do a lease renewal) to show its availability [9].

Although the service items are stored in a flat manner, a wide variety of hierarchical views can be imposed on the collection by aggregating items according to service types and attributes. Lookup services use a template matching mechanism in its lookup process. Service items in the lookup service are matched using instances of the *ServiceTemplate* class [9].

```
public class ServiceTemplate implements Serializable {
    public ServiceTemplate(ServiceID serviceID,
                           Class[] serviceTypes,
                           Entry[] attributeSetTemplates) {...}
    public ServiceID serviceID;
    public Class[] serviceTypes;
    public Entry[] attributeSetTemplates; }
```

The constructor simply assigns each parameter to the corresponding field. The rule of matching is explained as following [9]:

A service item (*item*) matches a service template (tmpl) if:

- *item.serviceID* equals *tmpl.serviceID* or if *tmpl.serviceID* is null, and
- *item.service* is an instance of every type in *tmpl.serviceTypes*, and
- *item.attributeSets* contains at least one matching entry for each entry template in *tmpl.attributeSetTemplates*.

An entry matches an entry template if the class of the template is the same as, or a super class of, the class of the entry, and every non-null field in the template equals the corresponding field of the entry. Every entry can be used to match more than one template. For both service types and entry classes, type matching is based simply on fully qualified class names. Note that in a service template, for *serviceTypes* and *attributeSetTemplates*, a null field is equivalent to an empty array; both represent a wildcard.

As described here, the service template may include a reference to an array of Class objects. These Class objects indicate to the lookup service the Java type (or types) of the service object the client desires. The service template can also include a *service ID*, which uniquely identifies a service, and attributes, which must exactly match the attributes uploaded by the service provider in the service item. The service template can also contain wildcards for any of these fields. A wildcard in the service ID field, for example, will match any service ID.

3.2.3. Lookup service interfaces

Among the many Jini lookup services interfaces, the ServiceRegistrar and ServiceRegistration are two fundamental interfaces which provide means to enable interaction between services/clients and lookup services.

3.2.3.1. ServiceRegistrar interface

As aforementioned, at the end of a discovery process, the lookup service's proxy, an object named service registrar implementing *ServiceRegistrar* interface, is transferred to the service. Services then use this proxy to communicate with the lookup service to register

service, find other services, receive event notifications, and even to incrementally explore the collection of service items. A client who wants to find a service also obtains the service registrar object by discovering the lookup service.

The following shows the interface definition [9].

```
public interface ServiceRegistrar {
    ServiceRegistration register(ServiceItem item,
             long leaseDuration)throws RemoteException;

    Object lookup(ServiceTemplate tmpl)
        throws RemoteException;

    ServiceMatches
        lookup(ServiceTemplate tmpl, int maxMatches)
        throws RemoteException;

    int TRANSITION_MATCH_NOMATCH = 1 << 0;
    int TRANSITION_NOMATCH_MATCH = 1 << 1;
    int TRANSITION_MATCH_MATCH = 1 << 2;

    EventRegistration notify(ServiceTemplate tmpl,
                         int transitions,
                         RemoteEventListener listener,
                         MarshalledObject handback,
                   long leaseDuration)throws RemoteException;

    Class[] getEntryClasses(ServiceTemplate tmpl)
        throws RemoteException;

    Object[] getFieldValues(ServiceTemplate tmpl,int setIndex,
              String field)throws NoSuchFieldException, RemoteException;

    Class[] getServiceTypes(ServiceTemplate tmpl,
                        String prefix)throws RemoteException;
    ServiceID getServiceID();

    LookupLocator getLocator() throws RemoteException;

    String[] getGroups() throws RemoteException;

    }
```

The protocol for communications between the proxy and the lookup service is implementation-specific. All of the proxy methods adheres the Java RMI remote interface semantics unless explicitly noted. These methods have been defined [9]:

- ***register():*** the method is used to register a new service and to re-register an existing service. When a new service is first registered, the *item.serviceID* must be null. If *item.service* does not equal (using *MarshalledObject.equals*) any existing item's service object, then a new unique service ID will be assigned and included in the returned *ServiceRegistration*. If *item.service* does equal an existing item's service object, the existing item is first deleted from the lookup service (even if it has different attributes) and its lease is cancelled, but that item's service ID is reused for the newly registered item. To re-register an existing service, the *item.serviceID* is supplied as the one returned from initial registration. If there is a match of the service ID, the existing service item is first deleted (even if it has different attributes or a different service instance) and its lease is cancelled by the lookup service. Then the new one is registered under the same service ID.
- ***lookup()***: overloaded methods. The main difference between these two forms is the number of matches and service items that are returned. The one-parameter form returns the service object (i.e., just *ServiceItem.service*) from an item matching the template or null if there is no match. If multiple items match the template, an arbitrary matching service object is returned. The two-parameter form returns at most *maxMatches* items matching the template, plus the total number of items that match the template. The item array returned by the two-parameter form is a complete service item, which includes the service object, service ID, and all the attribute sets. The return value is never null, and the returned item array is only null if *maxMatches* is zero. For each returned item, if the service object cannot be de-serialized, the service field of the item is set to null and no exception is thrown.
- ***notify()***: The *notify* method is used to register for event notification. A client invokes the *notify()* method to register itself to receive events. A template of interested services and the transition types of interest are also passed to the method. A "*serviceEvent*" is sent to the registered client whenever a register, lease cancellation or expiration, or attribute change operation, results in an item changing its state in a way that satisfies the template and the transition type. The transition type is defined by a transition parameter which is a bitwise OR of the following three values, which are defined as constants in *ServiceRegistrar* [9]:

- *TRANSITION_MATCH_NOMATCH*: An event is sent when the changed item matches the template before the operation, but does not match the template after the operation (this includes deletion of the item).
- *TRANSITION_NOMATCH_MATCH*: An event is sent when the changed item does not match the template before the operation (this includes not existing), but does match the template after the operation.
- *TRANSITION_MATCH_MATCH*: An event is sent when the changed item matches the template both before and after the operation.

The *ServiceRegistrar* has three methods, called browsing methods, which let clients get information about registered service items. These three methods -- *getServiceTypes()*, *getEntryClasses()*, and *getFieldValues()* -- are called "browsing methods" because they enable clients to browse the services and attributes in the lookup service.

3.2.3.2. ServiceRegistration interface

The return of the method *register()* for a new registration is an instance of *ServiceRegistration*. A registered service item is manipulated by using a *ServiceRegistration* instance. *** up to here, this should be normal text format ***

```
public interface ServiceRegistration {
    ServiceID getServiceID();
    Lease getLease();
    void addAttributes(Entry[] attrSets)
        throws UnknownLeaseException, RemoteException;
    void modifyAttributes(Entry[] attrSetTemplates,
                          Entry[] attrSets)
        throws UnknownLeaseException, RemoteException;
    void setAttributes(Entry[] attrSets)
        throws UnknownLeaseException, RemoteException;
}
```

The methods are described as following:

- ***getServiceID()***: returns the service ID of this service.
- ***getLease()***: returns the lease for the service registration.
- ***addAttributes()***: adds the specified attribute sets to the registered service item. There should be no duplication of existing attributes.

- ***modifyAttributes()***: modify existing attribute sets. A number of rules have been defined for modifying attribute sets. As explained in [9], the lengths of the *attrSetTemplates* and *attrSets* arrays must be equal, or *IllegalArgumentException* is thrown. The service item's attribute sets are modified as follows. For each array index `i`: if *attrSets[i]* is *null*, then every entry that matches *attrSetTemplates[i]* is deleted; otherwise, for every non-null field in *attrSets[i]*, the value of that field is stored into the corresponding field of every entry that matches *attrSetTemplates[i]*. The class of *attrSets[i]* must be the same as, or a superclass of, the class of *attrSetTemplates[i]*, or IllegalArgumentException is thrown. If the modifications result in duplicate entries within the service item, the duplicates are eliminated.
- ***setAttributes()***: deletes all of the service item's existing attributes and replaces them with the specified attribute sets.

3.2.4. Lookup process

Once a service has registered with a lookup service via the join process, that service is available for use by clients who query that lookup service. To find a service, clients query lookup services via a process called lookup.

To perform a lookup, a client invokes the *lookup()* method on a service registrar object. A client, like a service provider, gets a service registrar through the process of discovery, as described in session 3.2.1. The client passes as an argument to *lookup()* a service template, an object that serves as search criteria for the query. The client may decide which overloaded form of *lookup()* method to use. The service template can include a reference to an array of Class objects. These Class objects indicate to the lookup service the Java type (or types) of the service object desired by the client. The service template can also include a service ID, which uniquely identifies a service, and attributes, which must exactly match the attributes uploaded by the service provider in the service item. The service template can also contain wildcards for any of these fields. A wildcard in the service ID field, for example, will match any service ID. The *lookup()* method sends the service template to the lookup service, which performs the query and sends back zero to many matching service objects.

The client gets a proxy of the matching service as the return value of the *lookup()* method. Then the client will interact with the service via the service proxy.

Commonly a client looks up a service by specifying a Java type, usually an interface. For example, if a client needs to use a printer, it would compose a service template that includes a Class object for a well-known interface to printer services. All printer services would implement this well-known interface. The lookup service would return a service object (or objects) that implemented this interface. Attributes can be included in the service template to narrow the number of matches for such a type-based search. The client would use the printer service by invoking on the service object methods declared in the well-known printer service interface.

As mentioned before, a Jini lookup service can register itself into another lookup service. By doing so, a number of Jini lookup services in a number of Jini systems can form a hierarchical lookup mechanism. The service items stored in each individual lookup service are not replicated to every Jini lookup service. Jini technology provides means for Jini lookup services to find each other by multicast mechanism. This may have some limitations due to firewalls and network configurations for multicast. Another way of federating Jini lookup services is via a federation manager which is a well-known Jini lookup service. Each local Jini lookup service registers into this federation manager. By querying the federation manager, each lookup service knows each other.

3.3. Resource Description Framework

The Resource Description Framework, RDF, is a W3C standard for describing Web resources. It provides a common framework for expressing metadata to make flexible statements about resources that are uniquely identified by Uniform Resource Identifiers (URI). RDF defines a specific XML-based language, referred to as RDF/XML, for representing RDF information and for exchanging the information between different computers. The form of

this information is defined by RDF Schema, the RDF vocabulary description language. It provides a way to define vocabularies to be used in RDF documents. RDF schema is extensible and evolvable over time by using a new base URI each time the schema is revised. RDF also defines a simple protocol and the RDF query language (SPARQL) to retrieve RDF information from a RDF registry. A specific registry, called Sesame, is presented in Section 3.3.3.

3.3.1. RDF data structure

RDF Information is expressed in RDF statements using triples consisting of subject, property (or predicate) and object elements. Such a triple is commonly written as P(S, O), where S is the subject which has a property (or predicate) P with values O. In a RDF triple, the subject represents a resource. An object represents a constant value such as a string or number, or it represents another resource. The property is the relation between the subject and the object. According to the RDF specification, an object can be the subject of another RDF triple with properties and values. Combining each triple together, a complex RDF graph can be created to fully describe a resource. RDF supports reification which means that any RDF statement itself can be the subject or object of a triple. This means RDF graphs can be nested or chained. Reification also provides a means to record information about when statements were made, who made them, or other similar information. In order to identify things globally and uniquely, RDF uses Web identifiers, called Uniform Resource Identifiers (URI) to identify the name of the subject.

A RDF/XML description of a P(S,O) statement starts with an XML element named <rdf:Description> which represents the subject S. The property P is an element that is nested in the <rdf:Description> element. The content of the nested element is the object O. The following shows an example of a RDF file serialized in XML format.

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:exterms="http://www.example.org/terms/">
```

```
    <rdf:Description rdf:about="http://www.example.org/index.html">
      <exterms:creation-date rdf:datatype="&xsd;date">1999-08-16
        </exterms:creation-date>
    </rdf:Description>
  </rdf:RDF>
```

The example describes a web page identified by the URI at "http://www.example.org/index.html" that has a property named "creation-date" with a value of "1999-08-16". More specifically:

- The first line of the RDF document is the XML declaration. The root element of the RDF document is **<rdf:RDF>**. It defines the XML document to be an RDF document. It contains a reference to the RDF namespace and other namespace references.
- The **xmlns:rdf** namespace, specifies that elements with the rdf prefix are from the namespace "http://www.w3.org/1999/02/22-rdf-syntax-ns#". It defines syntax and grammar for RDF.
- The **xmlns:exterms** namespace, specifies that elements with the exterms prefix are from the namespace "http://www.example.org/terms/".
- The <**rdf:Description**> element identifies a resource with the **rdf:about** attribute as the subject of the resource. The <rdf:Description> element contains elements that describe the resource as the resource properties. The prefix **extrems** of the element create-date means the element is defined in the "http://www.example.org/terms/" namespace. This name space is outside RDF and is not part of RDF. RDF only defines the framework. The element create-date has an attribute rdf:datatype which indicates that the object or the values of the property of that resource has a data type. In this example, the data type is a date as defined for XML schema.

RDF does not have any build-in set of data types of its own, such as integer, real, string, or date. Instead, RDF simply provides a way to explicitly indicate what data type should be used. The data types are defined outside of RDF, and identified by their URIs. RDF recommends using data types defined for XML schema as its data types.

RDF also does not have any build-in vocabularies to describe the resource properties except the basic framework used to define the syntax and grammar. RDF encourages using common well-defined vocabularies such as the Dublin Core Metadata Initiative [18] for the purpose of achieving interoperability among machines.

3.3.2. RDF schemata

The RDF Schema (RDFS) is a mechanism which provides a basic framework to let developers define a particular vocabulary for RDF data and specify the kinds of objects to which predicates can be applied. RDFS does this by pre-defining some concepts, such as Class, subClassOf and Property, which can then be used in application-specific schemata. These application-specific schemata define classes representing groups of resources. Properties are also defined in the schemata and are associated to classes. In other words, RDFS does not provide a vocabulary of application-specific classes and properties; instead it provides the facilities needed to describe such classes and properties, and to indicate which classes and properties are expected to be used together. The following is a simple example of a RDF schema:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:base="http://www.science.uva.nl/research/sne/ndl#">
        <rdfs:Class rdf:ID="Location" />
        <rdfs:Class rdf:ID=" Device" />
        <rdf:Property rdf:ID="locatedAt">
            <rdfs:label xml:lang="en">located at</rdfs:label>
            <rdfs:domain rdf:resource="http://www.science.uva.nl/research/sne/ndl#Device" />
             <rdfs:range rdf:resource="http://www.science.uva.nl/research/sne/ndl#Location" />
        </rdf:Property>
        <rdf:Property rdf:ID="name">
            <rdfs:label xml:lang="en">name</rdfs:label>
            <rdfs:comment>A short human-readable name for the subject.</rdfs:comment>
            <rdfs:domain rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource" />
            <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
        </rdf:Property>
</rdf:RDF>
```

In this example, the RDF schema framework, located at http://www.w3.org/2000/01/rdf-schema# is used to define vocabularies that can be used for defining application-specific schema. The RDFS concepts have a prefix of rdfs. Two application-specific classes are defined: “Location”

and “Device”. The element <rdfs:Property> is used to define two properties: “locatedAt” and “name”. In the definition of “locatedAt”, rdfs:domain specifies that the property “locateAt” is applied to the class Device. And rdfs:range defines that the property “locatedAt” has a value which is an instance of the class Location. The property “name” is applied to the root class Resource which indicates that the “name” property is used for each RDF class, since Resource is the super-class of everything. The value of the name property is an instance of the class Literal. RDF schemata expressions are also valid RDF expressions. In fact, the only difference with ‘normal’ RDF expressions is that, in RDF schema statements, the semantics of certain terms is defined which is important for the interpretation of certain statements.

The above mentioned RDF schema is supposed to be located at www.science.uva.nl/research/sne/ndl.

An example of a RDF description of a device named TORO1OME1 which is located in Toronto is described as follows:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmls:ndl="http://www.scinece.uva.nl/research/air/ndl#" >
        <ndl:Location rdf:about="#Tornoto">
                <ndl:name>Toronto</ndl:name>
        </ndl:Location>
        <ndl:Device rdf:about="#TORO1OME1">
                <ndl:name>TORO1OME1</ndl:name>
                <ndl:locateAt rdf:resource="#Toronto" />
        </ndl:Device>
</rdf:RDF>
```

In this device.rdf document, one can see that, following the RDF schemata, the class Device has the name “TORO1OME1” and is located at location “Toronto”. Because the location is a resource, RDF makes it an attribute of <ndl:locateAt> specified by rdf:resource.

3.3.3. Querying RDF databases

RDF documents are usually stored in a registry called a RDF store. The detailed implementation of a RDF store depends on the underlying technology. In order to retrieve

information from a RDF store, the Simple Protocol and RDF Query Language (SPARQL) is defined by W3C as a standard query language and data access protocol for RDF. The SPARQL specifications define a simple protocol which provides a means of conveying SPARQL queries and returning results between query clients and query processors. The SPARQL protocol is described abstractly in a WSDL file called "protocol-query.wsdl" (SPARQL WSDL) [21] which defines types, binding information and an interface. Any application that has implemented the interface conforming to the SPARQL WSDL description can provide a RDF query service. The SPARQL protocol has only one interface, *sparqlQuery*, which in turn contains only one operation called *query* which is used to convey a SPARQL query string.

A SPARQL query string is a tuple (GP, DS, SM, R) where:

- GP is a graph pattern which is used for template matching in the RDF store.
- DS is a RDF dataset which represents the RDF data in the RDF store. The DS is an option in a SPARQL query.
- SM is a set of solution modifiers which specifies how to arrange the returned result. The following modifiers are used: ORDER BY, PROJECTION, DISTINCT, OFFSET and LIMIT.
- R is a result form which uses the result from pattern matching to form the result sets or RDF graphs. The defined result forms the following [22]:
  - SELECT - Returns all, or a subset of, the variables bound in a query pattern match.
  - CONSTRUCT - Returns an RDF graph constructed by substituting variables in a set of triple templates.
  - DESCRIBE - Returns an RDF graph that describes the resources found.
  - ASK - Returns a boolean indicating whether a query pattern matches or not.

The SPARQL query language is a SQL-like language for RDF. One would see the similarity from the use of the terms of SELECT, FROM and WHERE in both SPARQL and SQL queries. The SPARQL uses a simple syntax to specify variables and triplet templates for retrieving information from a RDF store. Most forms of SPARQL queries contain a set of triple templates called a *basic graph pattern*. Triple templates are like RDF triples except that each of the subject, predicate and object may be a variable. A basic graph pattern *matches* a sub-graph of the RDF dataset when RDF terms from that sub-graph may be substituted for

the variables, and the result is a RDF graph equivalent to the matched sub-graph [22]. An example of a simple query is shown in the following:

```
PREFIX ndl: <www.science.uva.nl/research/sne/ndl>
SELECT ?device
FROM <device.rdf>
WHERE { ?device ndl:locateAt ?location.
          ?location ndl:name "Toronto". }
```

This example is based on the RDF triples described in device.rdf mentioned in section 3.3.1. The keyword PREFIX is essentially declaring a namespace. A query can include any number of PREFIX statements. The query is to select devices represented by the variable ?device that satisfy the constraints given in the WHERE clause. The constraints use one additional variable ?location. The device variable must be located at a location whose name is "Toronto". This query will return all devices that are located in Toronto.

SPARQL allows for more complex queries by using SPARQL build-in functions and operators. For example, one can search devices that are not located in Toronto. The query is shown as follows:

```
PREFIX ndl: <www.science.uva.nl/research/sne/ndl>
SELECT ?device
FROM <device.rdf>
WHERE { ?device ndl:locateAt ?location.
          FILTER (?location != "Toronto") }
```

In the example, the term FILTER indicates that these results which are not located in Toronto will be returned.

SPARQL also defines template and format for returned result. A full explanation of SPARQL query is out of the scope of this thesis. The SPARQL specification can be found at [22].

3.3.4. Sesame

Sesame [35] is an open source Java framework for storing, querying and reasoning with RDF and RDF schema. The Sesame framework architecture not only works as a repository for RDF data, but also provides a set of Java libraries for applications that need to work with

RDF data. Sesame's design and implementation are independent of any specific storage devices, thus, Sesame can be deployed using a variety of storage devices, including relational databases. Sesames supports concurrent control and provides a native support for RDF schema semantics through several query languages designed for RDF, such as SPARQL, SeRQL (Sesame RDF Query Language), RQL(RDF Query Language). The architecture of Sesame is shown in Figure 6.

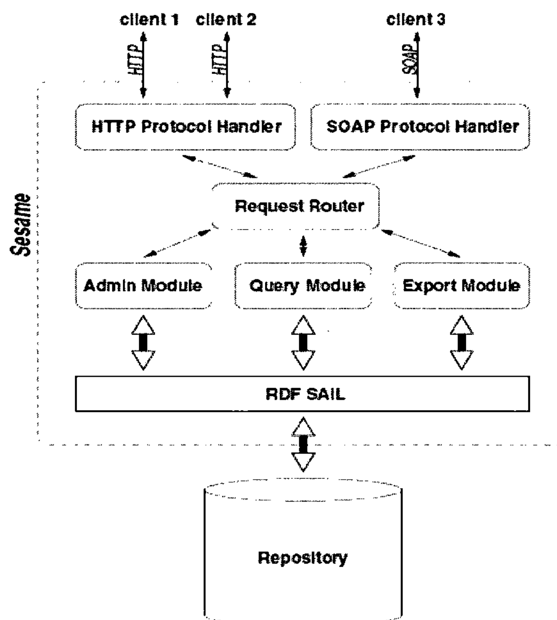

Figure 6: Sesame's Architecture [23]

Starting at the bottom, the RDF Storage and Inference Layer, or RDF *SAIL*, is an internal Sesame API that offers RDF-specific methods to its clients and translates these methods to calls to its specific repository. Such a repository can be implemented in a RDBMS, in memory, or in a file system. SAIL implementations can also be stacked on top of each other, to provide functionality such as caching or concurrent access handling [23]. It is the RDF SAIL module that allows Sesame to be implemented on top of a wide variety of storage repositories without changing the other Sesame components.

On top of the RDF SAIL are Sesame's functional modules. The admin module provides functions for managing the Sesame system, such as configuration and user account

management, uploading/deleting RDF data from the repository and so on. The export module is able to export the contents of a repository in the form of XML-serialized RDF files. The idea behind this module is to achieve interoperability between Sesame and other RDF tools. The query module provides query engines for different query languages such as SPARQL, SeRQL and RQL. Each query will be parsed and optimized by the query engine and translated into a set of calls to the RDF SAIL module. The returned data from the RDF SAIL module is also processed by the query engine to form the desired query result.

Client applications access the Sesame modules via the protocol handlers. Depending on the environment in which the Sesame is deployed, different ways to communicate with the Sesame functional modules are available. For example, communication over a browser interface may be preferable in the web context; however, in other context protocols such as Remote Method Invocation (RMI) or SOAP may be more suitable. In the Sesame system, protocol handlers are provided as intermediaries between the functional modules and their clients, each handling a specific protocol. The clients' requests to the functional modules are classified and routed via a request router module. This module provides a simple function to route client requests to the admin, the query engine or the export model according to the nature of the request. The returned results are also routed by the request router to the appropriate handlers.

Chapter 4 - User-Controlled Lightpath Provisioning System

User-controlled lightpath provisioning system, version 2, called UCLPv2 [33], is a software solution that enables network users to fully manage their own network resources to establish end-to-end lightpaths across multiple network domains without the interference of network operators. A lightpath in an optical network is defined as a point-to-point optical connection based on wavelength division or time division multiplexing. It is a dedicated communication channel and has a guaranteed bandwidth. In traditional optical network, network resources are managed by network operators. Setting up an optical connection to send traffic from one end to the other is a manual and time-consuming process. Upon user requests, network operators use a network management system (NMS) to configure network elements to configure the connection. It is not unusual for the turnaround time for a new connection to take up to six weeks to configure after the initial request has been submitted [12]. It is acceptable for a provision to take weeks or months for a connection which may be in place for long periods of time. However, this is unacceptable for research projects such as sensor networks, Grid computing, E-science applications and high-definition video-on-demand streaming applications that require dedicated on-demand connections for intensive data transmission, as well as dynamic provisioning from time to time. Consequently, a user- controlled lightpath provisioning system was envisioned and implemented [25]. UCLPv2 abstracts network elements, links and interfaces into network resources represented by software objects. It empowers users to control these objects to create/delete end-to-end lightpaths, and partition a lightpath into smaller capacity lightpaths. UCLPv2 also provides an interface to let third-party applications create lightpath connection by invoking UCLP functions through a Web Service interface.

One of the key concepts of UCLPv2 is APN (Articulated Private Network) [33]. In a physical network, an APN comprises a collection of network resources such as lightpaths and interfaces, as well as the network equipments connected to the lightpaths and interfaces. The network resources of an APN are assigned to a network user or an organization. This is similar to a

virtual private network. By abstracting the physical network resources into logical software objects the physical APN is virtualized into a logical APN and managed by the UCLP system. One of the advantages of virtualization of a physical APN is that its network topology can be changed or articulated dynamically by the end-users of the UCLP system [25]. The network topology refers to the logical configuration of one or more end-to-end connections using resources within an APN. Users can define different network topologies in APN scenarios. Each APN scenario is implemented as a collection of Web Service resources. By executing an APN scenario, end-to-end lightpath connections are created. UCLPv2 not only provides bandwidth-on-demand function, but it also provides network infrastructure services through Web Service technologies.

Another key concept of UCLPv2 is the fundamental lightpath. A fundamental lightpath is defined as an abstract representation of the basic unit of optical network partitions with a dedicated communication channel between two adjacent network elements. Depends on the technology underlying the network, it may represent a SONET channel, a wavelength in a Wavelength Division Multiplexing (WDM) network and so on. The fundamental lightpath hides the technology detail by abstraction, thus making it technology-independent. It is especially beneficial when making an end-to-end connection across heterogeneous networks. Furthermore, a lightpath can be partitioned into several smaller-capacity lightpaths following certain constraints. Several lightpaths also can be bound together to form a super-lightpath. By modeling physical resources as software objects, the system enables users to provision lightpaths based on physical links.

UCLPv2 is a service-oriented system in which the physical network resources such as switches, links and interfaces, are abstracted into Web Service resources and managed through Web Services. As a result, users can easily manipulate these resources through the use of Web Service without knowing the underlying complexity of the network technology. Each system module provides functionalities and interacts with others through Web Services. Thus, UCLPv2 system provides an architecture that lets third-party applications easily integrate UCLPv2

functions to provide high-level services or applications, such as advanced reservation system, network management and visualization systems and so on.

As aforementioned, an APN can be considered as a physically isolated or "underlay" network where a user can create a customized multi-domain network topology by binding together layer1 to layer3 network links, computers, time slices and virtual routing/switching nodes. This UCLPv2 system capacity is realized by representing all network elements, devices, links and interfaces as Web Service resources and binding various Web Service resources together to form an APN. Users can define APN scenarios by defining different connections. Each APN scenario represents a network topology consisting of multiple connections. By executing the APN scenario, users can dynamically change network topologies to suite their application requirement.

The UCLPv2 system management domain is divided into administratively independent organizations. An organization is an entity that owns a group of physical network resources and/or logical resources. There are three types of users in an UCLPv2 system: physical network administrator, organization administrator or APN administrator, and end-user. Different users have different levels of privileges. A physical network administrator belongs to an organization that owns physical networks. The job of the Physical Network Administrator is to initialize the system on a physical network and allocate resources from that network to be used in UCLP and to assign subsets of the resources to different organizations. An APN administrator belongs to an organization that does not own physical networks but acquires some logical resources from a physical network provider. He or she is responsible for managing and provisioning the resources for the purpose of building APNs. A typical operation is to create multiple APNs and assign them to different end-users. Once all the resources have been allocated and organized, the end-users can use them to create end-to-end connections by executing APN scenarios. An end-user can also re-configure the APN to meet the requirement of the applications.

4.1. Example of use of a UCLPv2 system

The UCLPv2 system has been used within CANet4. CAnet4 is the Canadian nationwide research and education network owned and operated by CANARIE Inc., with major funding of its programs and activities provided by the Government of Canada. Its main goal is to facilitate the development and use of Canada's advanced communications infrastructure. One of the characteristics of CAnet4 is that it cannot be considered as a single network, but as many parallel networks running over the same physical substrate. This substrate is an aggregation of point to point 10 Gbps wavelengths from different carriers.

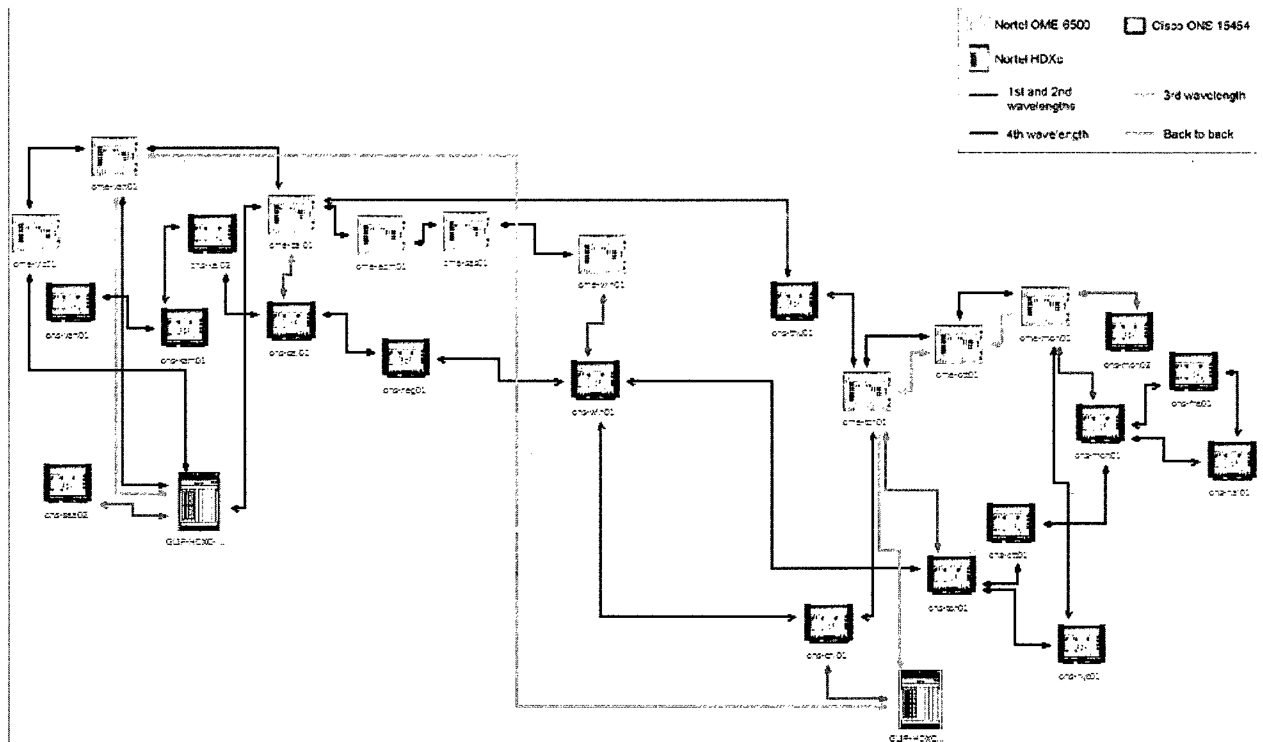

Figure 7: CAnet4 physical network view in the UCLPv2 system (screen-shot)

As depicted in Figure 7, CAnet4 currently has 4 wavelengths terminating on three kinds of network elements: 9 Nortel Optical Multiservice Edge 6500 platforms, 2 Nortel HDXC platforms and 16 Cisco ONS 15454 Multiservice Transport platforms. This SONET based equipment can offer Gigabit Ethernet or 10 Gigabit Ethernet WAN PHY services to its users.

In order to use UCLPv2 on CAnet4 network, the CAnet4 physical network administrator has created a view of the physical network in the system reflecting the real CAnet4 networks using the UCLPv2 client GUI. The physical network view consists of graphical icons representing network elements (NEs) for different optical network switches. Upon the creation of these network elements, a polling mechanism pulls the equipment's information from the physical network elements. The information is kept inside the UCLPv2 system and to be used for network resources abstraction. The network topology is represented by linking NEs together with lines which represent the physical fiber links between switches. This topology reflects the real physical network topology. Then the administrator will allocate network resources to be used in the UCLP system by creating logical resources including lightpathes and interfaces. These created logical resources are contained in a logical network view. The resources in the logical network represent the physical network resources being allocated to the UCLPv2 system. Figure 8 shows this logical UCLP network for CAnet4.

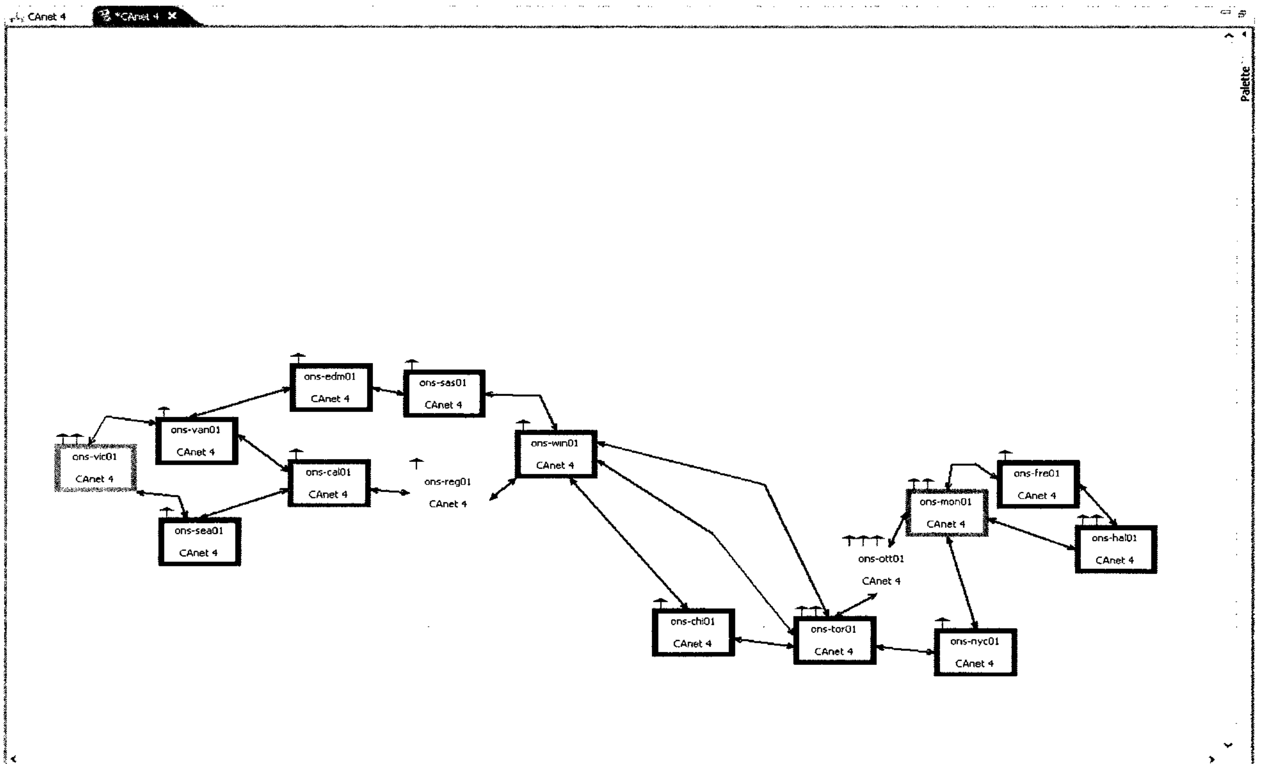

Figure 8: Logical network view of CAnet4

The logical network consists of arrows and links representing interfaces and lightpaths, respectively. One of the users of CAnet4, NRC (National Research Council Canada) has requested to interconnect their laboratories and institutes across Canada via CAnet4 dedicated lightpaths. Consequently, a subset of the lightpaths and interfaces are grouped into a resource list and exported to the organization of NRC. The resource list is an XML file describing these logical resources, and it can be sent to the APN administrators of NRC by any electronic means. The APN administrator of NRC launches the client GUI which 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 9.

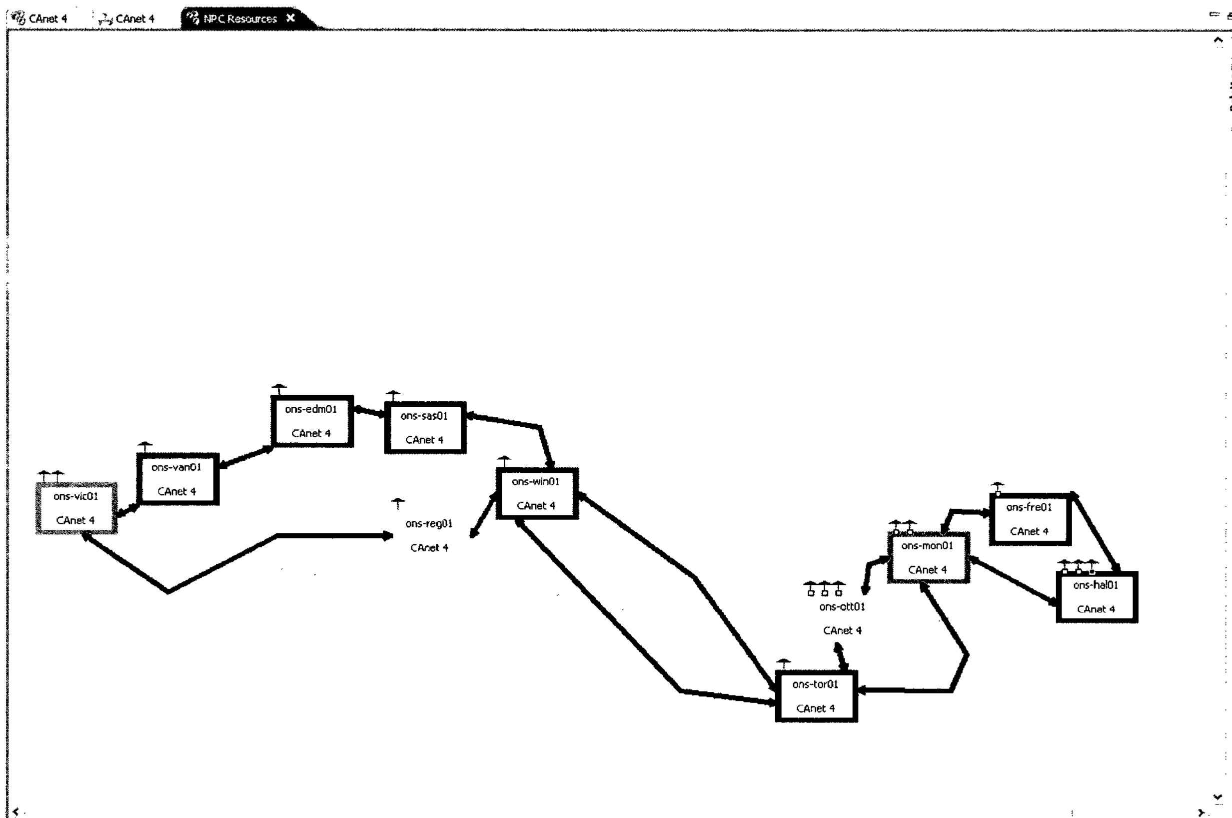

Figure 9: Resource List for NRC

In the NRC resource list, the links represent lightpaths between nodes that have a bandwidth of 1Gbps. The NRC administrator can then partition each 1Gbps lightpath into smaller bandwidth lightpaths to meet the inter-connection requirements while utilizing the resources more efficiently. The partition decision is based on the real bandwidth requirement of each inter-

connection and the applications that will run on the connections. The NRC administrator also can harvest resource lists from other networks, such as provincial research and education networks to form a multi-domain APN and extend the connection to other remote sites. Different inter-connections will be defined in different APN scenarios according to NRC's end users and their application requirements.

The NRC end-user uses the client GUI to view, configure or execute an APN. Execution of an APN means making the cross connections on the switches along end-to-end paths. Figure 10 shows a NRC APN with ten established end-to-end connections. After activating the topology by executing the APN scenario, data transfer between any two inter-connected end points can take place. In case of new applications that require different end-to-end connections, the user can stop the APN scenario, change its topology, and re-execute it.

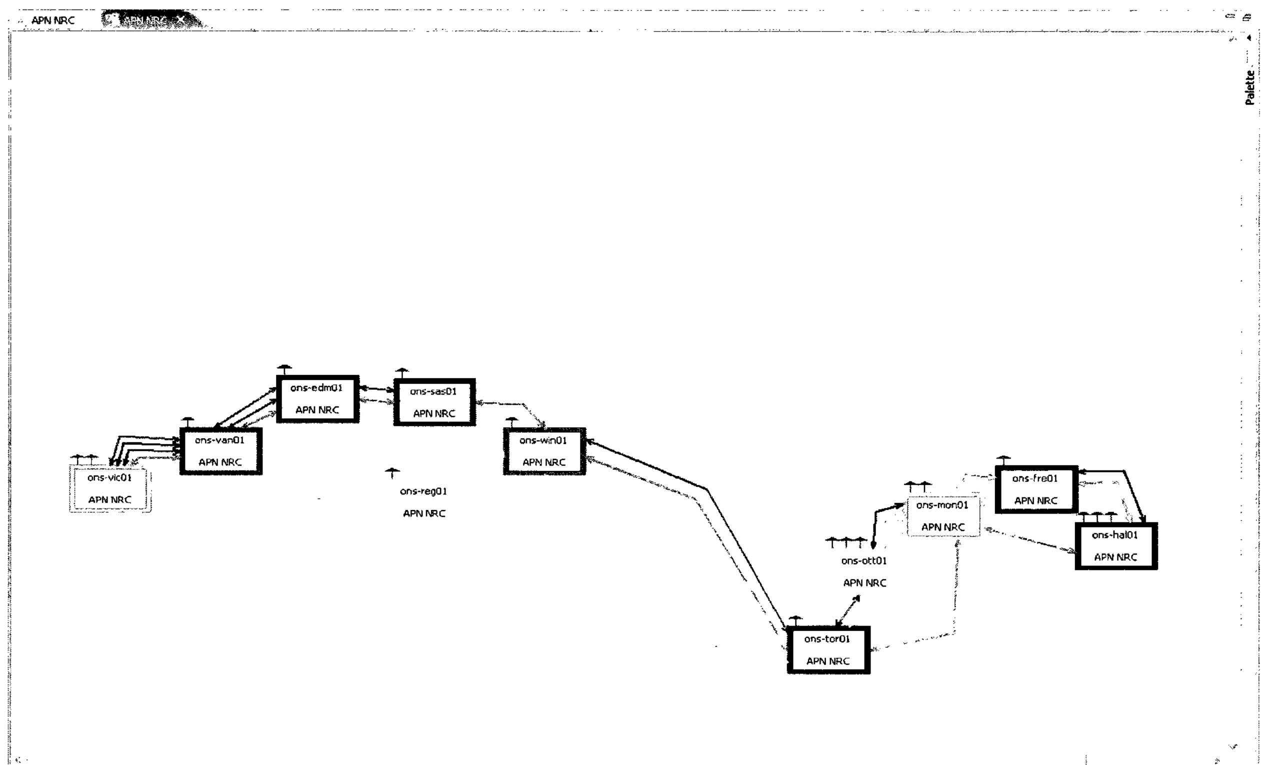

Figure 10: NRC APN

4.2. UCLPv2 system architecture.

The design and implementation of the UCLPv2 system is based on SOA and utilizes multiple technologies including Eclipse Rich Client Platform (RCP), Web Services, Web Service Resource Framework (WSRF) and Global Toolkit4 (GT4). Java is the programming language chosen. The architecture of the UCLP system consists of three layers: Higher level Services/Application layer, Resource Virtualization Service Layer, and Resource Management Service Layer. Figure 11 shows the high-level architecture of the system.

4.2.1. Resource Management Service Layer

The Resource Management Service is a group of Network Element Web Services (NE-WS) that control a family of physical network elements such as optical switches. Each NE-WS is dealing with a different technology and behaves like a proxy that sits on top of one or more network elements. 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 WSDL interface of the service has two port types. The configuration port type contains methods to add, delete, and modify the configuration parameters of the network element to be managed. The operation port type provides different operations to cross-connect and uncross-connect on a switch. The service translates XML requests into TL1/CLI commands in order to control physical devices that use the standard TL1 interface. 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 [26].

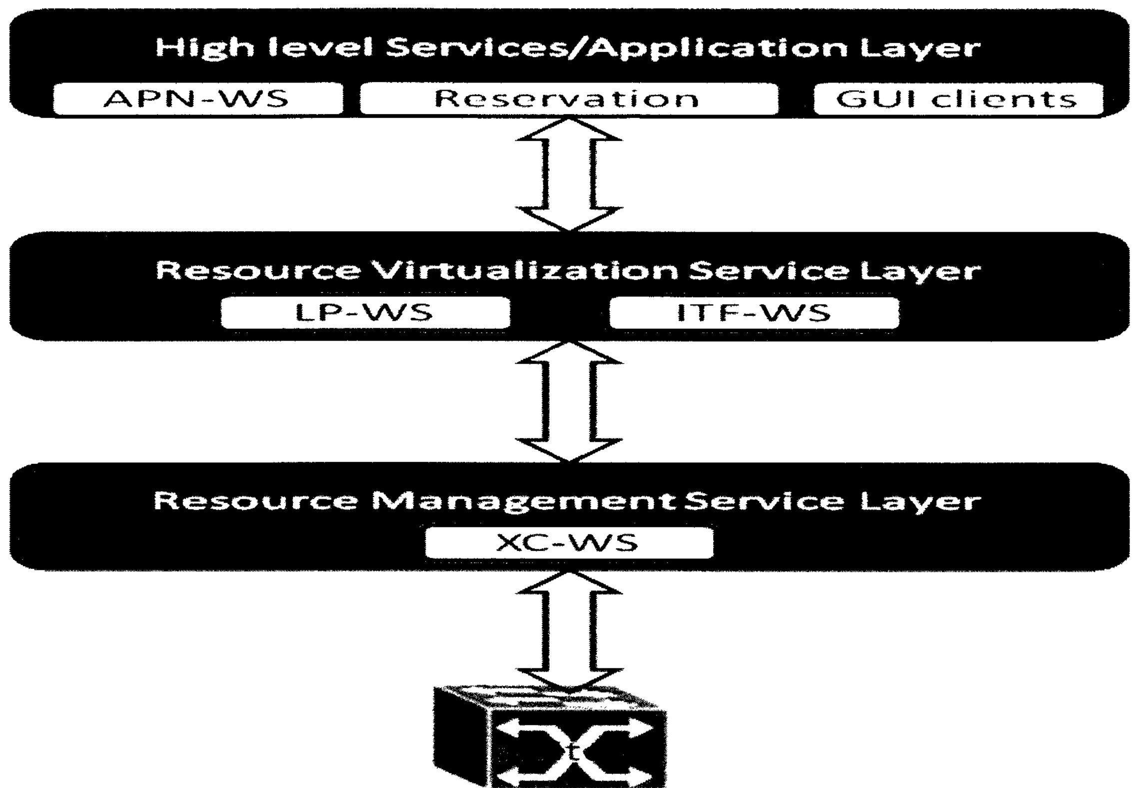

Figure 11: UCLPv2 Service Oriented Architecture

4.2.2. Resource virtualization service layer

The objective of the resource virtualization service layer is to provide an abstracted view of the lower-layer network resources. The idea is to provide a layer of abstraction between higher level user-defined services and the NE-WS. As a result, the details of invoking the lower layer services become transparent to end-users and applications. Two fundamental core Web Services have been defined in this layer to abstract network resources into Web Services: the Lightpath Web Service (LP-WS) and the Interface Web Service (ITF-WS). As introduced in the previous chapter, WSRF allows a Web Service to implement the factory pattern and to expose and manipulate the state of a Web Service in a standard fashion through the use of the Web Service resource abstraction, WS-Resource. Each individual piece of state of a Web Service is called a resource, and it is identified through the use of a Web Service Addressing (WS-A) EndPoint Reference (EPR) [29]. This EPR is composed by the URL of the Web Service plus a key that uniquely identifies the resource within the Web Service. The LP-WS and ITF-

WS are implemented as a factory service and an instance service complying with the WSRF specifications. The LP-WS factory service is responsible for creating new lightpath resources (individual lightpath, super-lightpath and other connections) and operations to query, or bond lightpath resources. The LP-WS instance service has the responsibility to perform the following operations on a lightpath resource: to destroy a lightpath resource, to partition it, to set endpoints or to change the owner. ITF-WS provides the same functions from its factory and instance services, except the "create_super" operation.

A lightpath is a reserved, private, communication link between two network ports on two network optical switches. It has two endpoints which represent the two ports linked by the lightpath. The type of the ports can be SONET/SDH, DWDM, Ethernet or MPLS. Consequently, a lightpath can represent time slots connecting two SONET ports, or a wavelength linking two DWDM ports, or an Ethernet channels. By abstracting into a Web Service resource, a lightpath logically represents a communication link connecting two endpoints. The Web Service-enabled endpoints are in the form of WS-Resources. Each endpoint WS-Resource consists of two parts: an EPR and the state of the endpoint in the lightpath Web Service. The EPR consists of an URL pointing to the lightpath Web Service (LP-WS), and a unique key to identify the resource within the LP-WS. The two endpoints WS-Resources together form the lightpath resource. The state of the lightpath represents the lightpath properties such as the owner of the lightpath, and the bandwidth and so on. Users access a lightpath resource by invoking its Web Service pointed by the endpoint EPRs.

An interface is a single network port on a network element. It can be an Ethernet port or a SONET/SDH or a DWDM add/drop port. An interface is abstracted into a WS-resource called interface resource, which has an EPR and the resource represents the interface states. Figure 12 shows UML class diagrams of the lightpath resource and interface resource.

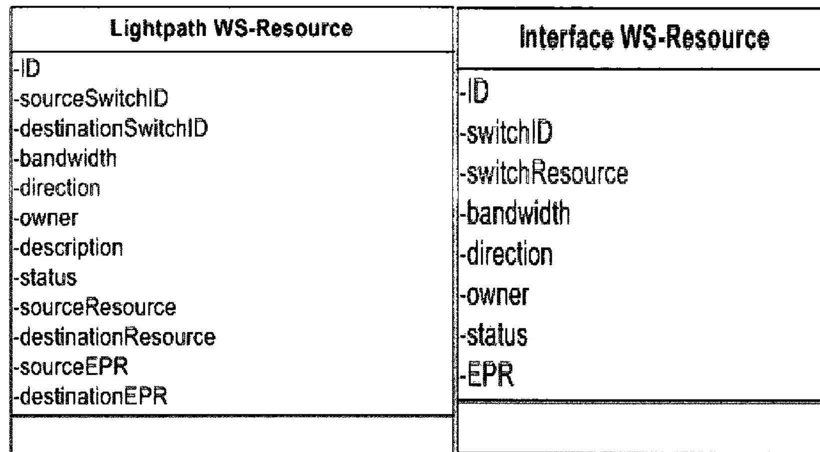

Figure 12: UML class diagram of Lightpath and Interface resource

The lightpath/interface resources and the LP-WS, ITF-WS provide a virtualized view of a network in the form of a network composed of the lightpath and interface resources. All the detailed technology specifications are isolated from the users and their applications by the Web Service resource abstraction. Only high-level parameters, like bandwidth, delay, jitter or start/end time are exposed to users. The abstraction is technology independent which makes it very easy to be manipulated and integrated into user applications.

4.2.2.1. Resource lists

A resource list, RL, is formed by putting virtual nodes, lightpath resources, interface resource and, or connections together into a container called an Articulated Private Network (APN) container. A RL represents a partition of a physical network. The UCLPv2 system shares resources via RLs among organizations. Figure 13 shows a class diagram of a resource list.

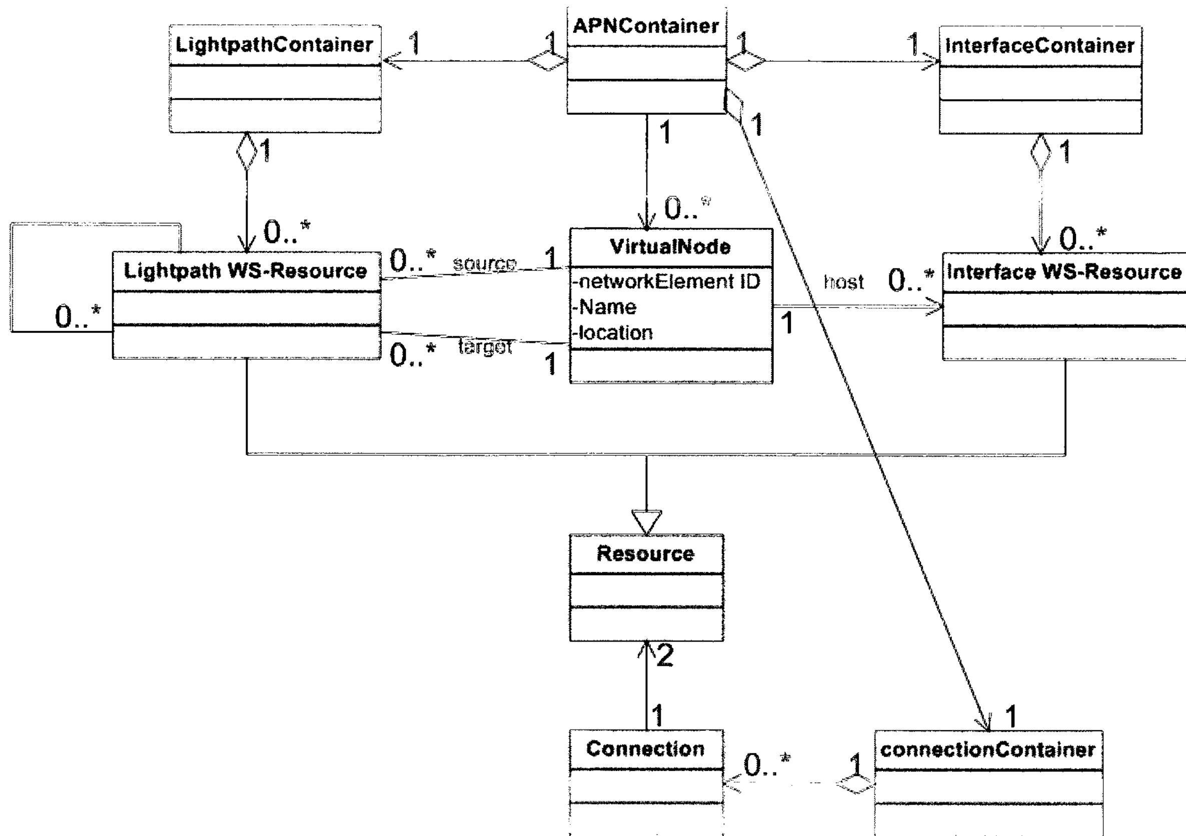

Figure 13: UML class diagram of Resource List data structure

As shown in the Figure 13, a RL is represented by an *APNContainer* with three sets of resources: interface resources, lightpath resources and connections. Each set of resources is grouped by a wrapper class (*InterfaceContainer, LightpathContainer* and *ConnectionContainer*). The *APNContainer* contains zero or more virtual nodes which represent a partition of a physical node in a physical network. Each virtual node can have zero or more interface resources, source lightpath resources (a partition of a physical link) and target lightpath resources. The Resource class contains interfaces and lightpath resources referring to these resources in lightpath and interface containers. A lightpath resource can also represent a super-lightpath resource if it contains one or more lightpath resource (a super-lightpath is a concatenation of two or more lightpaths). Finally, a connection contains two or more connected resources. When a RL is being created, all the

containers are having zero resource. This means the RL is an empty resource list and has to wait for the admin user to enter resources.

In order to form a resource list and include the necessary network topology information, the UCLP system uses the concept of virtual nodes to represent physical network elements. A virtual node is an abstraction of a physical network element that contains different endpoints referenced by lightpath and interface resources. In other words, a virtual node is the anchor point of lightpath and interface resources. A virtual node contains link information to its adjacent nodes. A Web Service-Addressing (WS-A) format Endpoint reference point (EPR) is contained in the node to indicate the Web Service access point of the XC-WS of the network element that is represented by the virtual node. The following is an example of a virtual node in XML format with a RL document:

```
<vn dmode="NORTEL OME 6500" physicalNetwork="CANARIE NET" parent="CANARIE HPDMnet RL1"
      name="OTWA2OME1" lx="378" ly="907" w="84" h="73" >
    <svlp>CANARIE NET-OTWA2OME1-1-5-1-TORO1OME3-1-11-1</svlp>
    <svlp>CANARIE NET-OTWA2OME1-1-6-1-MTRL2OME1-1-6-1</svlp>
    <tvlp>CANARIE NET-OTWA3OME1-1-14-1-OTWA2OME1-1-11-1</tvlp>
    <epr
      uri="https://uclp02.canarie.ca:8443/wsrf/services/argia/OpticalSwitchService"
      keyName="OpticalSwitchKey"
      keyNamespace="http://argia.inocybe.ca/2007/10/opticalswitch"
      keyHost="uclp02.canarie.ca" port="8443" key="a40ffca0-2de7-11de-8046-d378f9839ee0"/>
</vn>
```

In this example, *<svlp>* elements indicate that the node OTWA2OME1 has links to nodes called TORO1OME3, MTRL2OME1 where the node OTWA2OME1 is the source of the links; whereas the *<tvlp>* indicates the node has a link where TWA3OME1 is the target. The words *source* and *target* represent a link's direction. The *"svlp"* and *"tvlp"* are the short forms of "source virtual link persistent" and "target virtual link persistent", respectively. The *<EPR>* element represents the endpoint reference which contains the URI of the XC-WS Web Service and the key as the identification of the state of the node in the XC-WS. The lx, ly, w and h attributes stand for *locationX*, *locationY*, *width* and *height*, respectively. These attributes are for GUI displaying purpose.

In the UCLPv2 system, Castor [24] is used to map a RL from Java objects into an XML document. The Castor is an open source Java library for converting Java objects into XML documents according to pre-defined map files. An XML formatted RL is suitable for transfer between different organizations.

4.2.3. High-level Services/User applications

The UCLPv2 resource virtualization layer provides an abstracted view of the network. Heterogeneous networks can be viewed as a pool of lightpath and interface resources with core Web Services. The resources can be partitioned, bonded, subleased or connected by invoking the operations of the core Web Services regardless of the underlying technology. The resource abstraction enables users to build their network services/applications on top of the resource virtualization layer. The following introduces two such applications: a resource management center (RMC), and an advance reservation system, Chronos.

4.2.3.1. RMC

The Resource Management Center (RMC) is the Graphical User Interface used to manage the physical infrastructure, the network resources and the APNs of an organization in The UCLPv2 system. It is built on top of Eclipse's Rich Client Platform (RCP) and consists 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 [12].

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 a physical network representation, provision network elements into logical resources such as lightpath and interface Web Service resources. The resource list editor enables physical network administrators and APN

administrators to view, edit and manage their logical resources. Admin users can create new resource lists from existing resources or create 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 APN Web Services to establish end-to-end connections. Once logical resources are created, they are stored both on the local computer where the GUI is running and a remote database for backup purposes. These resource objects not only serve as a basis for graphical display, they are also used as resource identifiers when communicating with the core Web Services. An example of using RMC GUI is shown in Section 4.1.

The RMC stores the physical networks, logical networks, and resource lists, plus other graphical information (like the colors, locations in the map, line styles, background pictures) into a workspace. A workspace is a feature of Eclipse. The RMC uses the workspace as follows: when the RMC starts, it creates a new project called "User Resources", which contains four folders: one for physical networks, another one for logical networks, another one for resource lists and the last one for custom APN resource lists. Each individual physical network, logical network, resource list and custom APN is stored as an XML file in the corresponding folder with its background image file for the GUI. The information kept in the workspace is local. To ensure that the information can be accessed and resources can be operated by other users or the same user via another computer, the local workspace information will be uploaded to the server each time the user logs out of the RMC. RMS also manages user account, user credentials and authentications.

4.2.3.2. Chronos

Chronos [32] is a reservation system that allows users to reserve network resources in advance or immediately to build end-to-end connections in the intra-domain, or inter-domains. It has a built-in routing function so that it is not necessary to know the inner topology of the network, users just select the endpoints to connect, the start and end times and the bandwidth needed, and Chronos finds a suitable path and reserves the resources

for that time. As a result, users can manage their network resources more efficiently and thus provide a better QoS. Furthermore, network domains controlled by Chronos/UCLPv2 can be plugged into the Harmony system [11], a software system developed inside the EU-IST Phosphorus Project that is able to create multi-domain reservations through different domains controlled by Chronos/UCLPv2, DRAC [14] and DARGON [15].

Chronos does not reserve the network resources. It just manages the reservations. When the reservation time is coming, Chronos will commission the connections via UCLP Web Services. UCLPv2 system administrators reserve resources for Chronos by creating a resource list with pre-defined name "Reservation". This "Reservation" resource list contains all the resources reserved for the Chronos system. Chronos needs to connect to the UCLPv2 database to download the resource list into its system. The resource list not only contains the network resources, but also contains network topology information by specifying the relations between network nodes, network links and interfaces in the resource list. Chronos is able to fetch network topology information from the resource list and find a suitable path between user- specified endpoints via its routing function. After all the needed lightpath and interface resources have been gathered along the path, the connection request will be sent to UCLPv2 connection Web Service according to the reservation type. After a connection request is sent to the UCLPv2 system, the UCLPv2 Connection Web Service will return a result to indicate whether the request was successful or failed. If the connection failed, Chronos will notify users via email and/or throw exceptions on the GUI if the GUI is open. In such a case, users need to contact the admin users of the Chronos and UCLPv2 systems to investigate the cause of the failure.

The user of Chronos may be a human, a grid middleware, or other types of applications. In general, two types of reservations can be created in Chronos: immediate reservations, which are made in a just-in-time manner, and advance reservations, which reserve resources to be used at the start time selected during the reservation. Actually, immediate reservations are a special case of advanced reservations where the start time is the present

time. Advance reservation is a useful network service. It is useful especially in any application where large amounts of data have to be transmitted over a network and the time of the transmission is known in advance. An advance reservation can be fixed or malleable: in fixed reservations the starting time and the duration are fixed, whereas in malleable reservations, both starting time and duration are variable.

The user may group several independent reservations into a single "job". This feature achieves a higher convenience when Chronos users plan complex workflows. While the workflow is not yet completely planned, these workflow items are blocked as "pre-reservations". Therefore, if the complete workflow was successfully planned, these pre-reservations can be modified to permanent ones. Chronos offers the possibility to create Draft Reservations which can be used as templates.

Chronos has two modules: a web client allows users to make, configure, delete reservations; an advanced reservation service module provides core functions of managing reservations and communication with UCLPv2. Figure 14 depicts the interaction between Chronos and UCLPV2: the end-users interact with the Chronos system to create the reservations via HTTP web page application, and Chronos calls UCLPV2 core Web Services to activate and delete the physical connections.

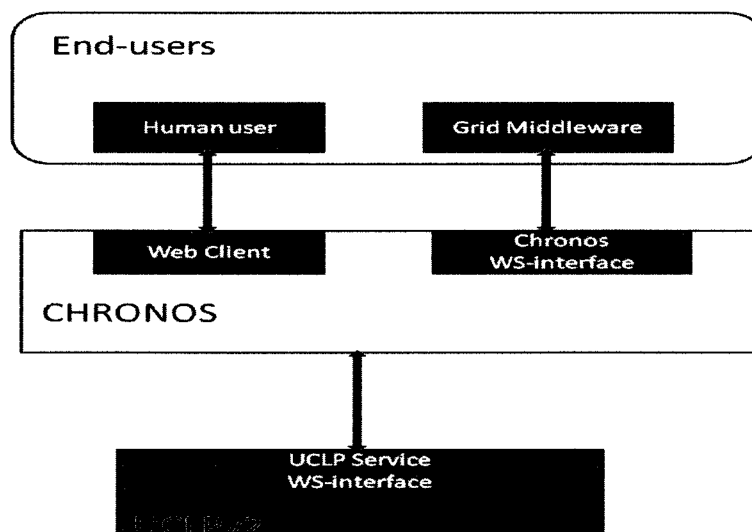

Figure 14: Interactions between Chronos

Chapter 5 - Designing resource advertising and discovery for UCLPv2

Organizations within the context of UCLP are independent administrative entities. UCLPv2 allows organizations to share their free resources by putting the resources owned by the organization into a resource list (RL) and exporting it to other organizations. The administrator of an organization can import resource lists exported by other organizations and create new RLs or APNs using the imported resources with or without the organization's own resources. The end-users utilize these resources to create end-to-end connections. Resource sharing is greatly beneficial for efficient resource utilization and inter-domain end-to-end connection creation. To create inter-domain connections, one organization needs to acquire the resources from other organizations by importing them in resource lists. All these resource acquiring processes involve the administrators of the organizations involved. The resource lists are usually transferred via email or FTP applications after the agreement has been achieved through some telephone conversations. This process of resource sharing is manual, usually error prone and inefficient. In order to automate this process, a suitable resource advertising mechanism is needed.

Another motivation for automating this process comes from the interoperability requirement for auto-provisioning among GLIF organizations. GLIF, *Global Lambda Integrated Facility*, is an international virtual organization of research networks, research consortia and institutions whose aim is to build a worldwide networking facility for scientific research. GLIF consists of a collection of optical exchange points, GOLEs (GLIF Open Lightpath Exchanges) and links between them. A GOLE node is an optical switch that works like a condominium switch in which the resources are shared by GLIF organizations. A global network is formed through lightpath connections to GOLEs. The global network currently consists of over a dozen GOLEs spread over North America, Europe, East Asia, and Australia, with numerous links across and between the continents [13]. In order to automate the process of provisioning lightpaths for various scientific projects, applications such as UCLP, DRAGON and DRAC have been developed and implemented to do provisioning and

brokering by different GLIF participants. DRAGON (Dynamic Resource Allocation via GMPLS Optical Networks) is a framework that can provision connections on heterogeneous networks. The DRAGON project is funded by the National Science Foundation of the USA as part of the Shared Cyber-infrastructure Division (SCI) Experimental Infrastructure Network (EIN) program. It provides support for provisioning connections using GMPLS, even for networking devices that do not support it. DRAC (Dynamic Resource Allocation Controller from Nortel) enables applications to request connections through the network. DRAC has been deployed in the Netherlight and SURFnet networks. UCLPv2 is implemented in the CANARIE network, Canada's research network. These applications are all aimed at providing network provisioning capabilities for single domains, or for multiple domains using the same application. Currently these applications cannot co-operate with each other for provisioning connections. Within GLIF the goal would be to achieve interoperability among these applications in order to support inter-domain brokering and provisioning directly or indirectly. One approach for supporting indirectly provisioning among these applications is provided by the Chronos system. Chronos is an advance reservation system which can reserve network resources to create end-to-end connections in UCLP, DRAC and DRAGON. As explained in Section 4.2.3.2, Chronos gets all the reserved resources from the UCLP system by downloading a resource list named "Reservation" from the UCLP server database. The whole process of reserving/acquiring resource from UCLP to Chronos is not an automatic one. The administrators of the UCLP system and Chronos need to co-ordinate closely in a manual process. In order to automate the process of reserving/acquiring resource for Chronos, a suitable resource advertising mechanism is needed in the UCLP system.

In this chapter, the description of UCLP resource sharing is presented in Section 5.1. Section 5.2 specifies the design requirements of resource advertising and discovering. Section 5.3 explains the design choices for designing a new mechanism of resource advertising and discovery within the context of UCLP v2 resources and resource lists. Section 5.4 describes the design in detail.

5.1. UCLPv2 resource sharing

UCLPv2 organizations share resources in the form of resource lists. In another word, a resource list, RL, is the base unit of conveying resources between organizations. After a RL is created by the administrator, it can be exported to another organization by changing the owner of the RL and its resources in the RL to the importing organization. UCLP v2 has made assumptions about exporting RLs: there is no super-lightpath, nor connection in an exported RL; the imported resources cannot be exported anymore. The exporting is done via the RMC client GUI function called "Export". After exporting the RL, it is transferred to the importing organization by email or FTP. The administrator of the importing organization imports the RL by putting it into the workspace of the UCLP RMC client running on his local computer and uploading it to the RMC. Then the imported RL can be manipulated as a native RL. By default, the lease end time is set to be last forever.

When it is time to release the RL, the importing organization needs to release the RL by changing the owner of the RL and its resources to the original owner. This is done via the RMC client GUI function "Release". By changing the ownership, the importing organization loses the visibility to the RL in the GUI. The RL's original organization also needs to change the ownership of the RL back to again control on these resources. All the processes of exporting, importing and releasing are manual and the administrators of the two organizations need to coordinate their job carefully to avoid possible errors.

5.2. System requirements

To facilitate the UCLPv2 organizations to share resources and automate the process, a suitable resource advertisement mechanism is required. In the current manual process of resource sharing, the requirements of the resources are known to both of the administrators of the exporting and importing organizations via their informal communications. However, we need a mechanism by which the importing organization can specify its requirements when resources are being advertised and discovered in a directory.

These resource requirements will not be known by a directory unless there is a sophisticated way of mapping the requirements to a set of resources that meet these requirements. A directory usually performs this mapping based on the formulation of the query, which consists of query terms, query grammars and functions, by composing the query strings with the stored resources. In the UCLPv2 system, users require resources such as interfaces and lightpaths to create their end-to-end connections. Such requirements are usually based on the resource properties such as location, bandwidth, type, round trip time etc. Consequently, the following basic requirements must be satisfied by the resource advertising and discovery mechanism for UCLPv2 systems:

a) UCLP users must be able to search a directory based on desired resource properties. The search may be based on a single property or a combination of the properties.
b) The search must not be limited to property-value matching by equality. It should also allow for range queries. For example, when a user search lightpath resources with a capacity of 1000Mbps, all the lightpaths that have a capacity equal or higher then 1000Mbps should be returned.

In addition to the above two basic requirements, the following requirements must be accommodated by the advertising and discovery system:

c) The implementation of a directory must be based on existing directory systems proposed for SOA systems. Therefore, UDDI, JINI and RDF/Sesame approaches will be considered in Section 5.3 and one of these systems will be selected as the basis for the implementation.
d) A resource list may either be publicly advertised which means that all the UCLP organizations can discover it, or it may be advertised to a particular organization which means that only the organization to which the resource list is advertised can discover the resource list. The latter is called reservation.
e) Advertised resource lists must have an end date to specify its lifetime in the directory. The end date of an advertised resource list can be extended by the admininstrtor of the advertising organization.
f) The system should allow UCLPv2 admin users to lease a resource list which is reserved for their organization or publicly advertised.
g) The directory should allow other systems such as Chronos, DRACK and DRAGON to discover resources.

5.3. Design choices for resource advertising

In this section, based on the system requirements, several service registries such as UDDI, the Jini lookup service and RDF directories are being considered and one of them will be selected for the resource advertising and discovery system. In order to achieve the inter-working with other systems, the NDL is also considered in this section.

5.3.1. UDDI

UDDI structures the information about the resources through four principle entities: the *businessEntity*, *businessService*, *bindingTemplate* and *tModel*. UDDI allows a wide range of searches of the registry: services can be searched by name, by location, by business, by bindings or by *TModels*. It is possible to look for all the services that have a given WSDL representation, or for all the services that adhere to the Rosetta Net specification. However, the search is limited to keyword matches in UDDI and UDDI does not support any inference based on the taxonomies referred to by the *TModels* [16]. Furthermore, a study [17] shows that UDDI is not suitable for resource discovery because of the following reasons: First, UDDI lacks explicit data typing. The data type of values in all the key-value pairs is limited to strings. Second, UDDI has no built-in notion of dynamic service information or any other mechanism in which the context of stored data changes over time. A resource state may be changed over time which indicates that the resource stored in the UDDI registry needs to be updated very often. However, UDDI does not support dynamic service data storage. In addition to the reason mentioned in [17], UDDI lacks a data model that can describe relations between different entities such as expressed by UCLP resource lists.

Some research [16] has focused on modifying the UDDI registry to add more meaningful metadata to enable semantic search and to express relationships among service entities. The result of such efforts leads to different data models such as DMALS, RDF/OWL to describe the services and mapping. The price is the increased complexity and reduced performance. We conclude that UDDI does not meet our requirements.

5.3.2. Jini lookup registry

Another choice is the Jini lookup registry. As described in Chapter 3, a lookup service maintains a flat collection of service items. Each item represents a proxy of a Jini Service. An item contains: a) the service proxy that clients use to access the service, and b) an extensible collection of attributes that describe the service or provides secondary interfaces to the service. UCLP's RLs can be perfectly represented by service entities with each attribute set represents the different resource container. Java classes can also be created to contain the metadata descriptions of an RL and store it in the Jini lookup service. Jini supports queries based on template matching which means finding service entities by service type and their attribute values. One rule based on the template matching is that the values of the attributes of a supplied template must match (by equality) the service item's corresponding attribute values. Jini lookup service is good for searching individual UCLP resources such as a lightpath or an interface resource which has an explicit type and is suitable for the template matching. This query model meets the requirement (a) as specified in section 5.1. However, Jini lookup service does not meet the requirement (b): It does not support queries such as finding lightpath resources whose bandwidth is greater or less than X. Such queries are useful for finding resources for creating an end-to-end connection by increasing the opportunity of finding all suitable resources; however, the Jini query model does not support such queries.

5.3.3. RDF/Sesame

The last choice is the RDF/Sesame approach. RDF is a W3C recommendation that was originally designed to standardize the definition and use of metadata-descriptions of Web-based resources. However, RDF is equally well suited for representing arbitrary data. As described in Chapter 3, RDF describes resources based on subject-predicate-object triplets, commonly written as P(S, O). That is, a subject S has a predicate (or property) P with value O. And the object O may be a subject of another triplet. The predicate P represents a relation between the subject and the object. This triplet representation is well suited to

describe the UCLP RLs and their resources. For example, a device is located at a place and it may contain interfaces. Interfaces have bandwidth and it may connect to another interface. All the relations between resources and resource properties can be easily expressed by RDF triplets. These relations represented by RDF triplets can be used as search templates for RDF queries to query UCLP RLs and resources stored in a RDF store. By replacing the subject, predicate and object with variables in the search template and combined with logical operations applied to these variables, SPARQL can easily meet the requirements of (a) and (b). Furthermore, RDF can divide the UCLP resources into groups as classes which, in essence follow the Java class paradigm. It is possible to define an application-specific RDF schema, in which the vocabularies for representing the subjects and various properties can be defined and the relations between the resource classes and their properties are also specified by associating properties to classes in the RDF schemata. By using the same schema, RDF resource descriptions can be understood by different RDF- enabled applications.

Sesame is an open-source Java framework for storing, querying and reasoning with RDF data and RDF schema. The Sesame framework architecture not only works as a repository for RDF data, but also provides a set of Java libraries for applications that need to work with RDF data. Sesame supports several query languages designed for RDF, such as SPARQL, Sesame RDF Query Language (SeRQL) and RDF Query Language (RQL). The UCLP system can use the Sesame and its Java libraries to create a repository for storing the RDF data and develop resource management functions for managing the stored RDF data.

5.3.4. Inter-working consideration

In order to achieve interoperability among UCLP, DRAC and DRAGON, the Network Description Language (NDL) [13] was developed to describe optical networks based on RDF. NDL consists of several RDF schemas in which vocabularies have been defined to describe optical network topologies, the network equipments and technologies. For example, NDL decomposes a network topology into RDF classes such as locations, devices, interfaces and

links; properties such as locatedAt, hasInterface, connectedTo and so on. These classes, properties and their associations are defined in the NDL topology schema which is illustrated in Figure 15. Since the schema is an XML file, the UML class diagram is used here to present the relationship among the XML elements. In the UML class diagram, the classes correspond to the RDF classes whereas the class attributes correspond to the RDF properties.

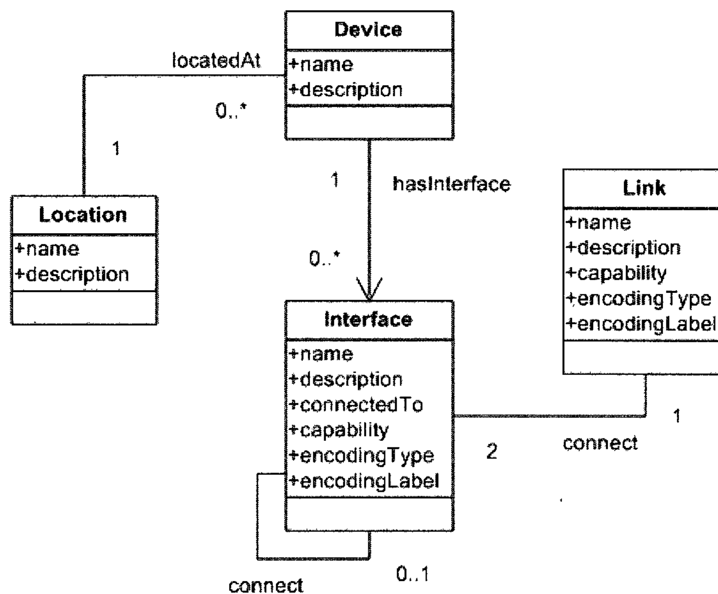

Figure 15: UML class diagram of NDL topology schema

As shown in Figure 15, the NDL topology schema conveys the following general information: A network device is located at a location and the device may have zero or several interfaces. An interface is connected to a link or it may be connected to another interface. A link connects two interfaces. An example NDL description of a simple network is shown in Figure 16. The NDL description is an XML file (listed in Appendix B). The UML instance diagram in the figure is used to show the XML elements and their relationships.

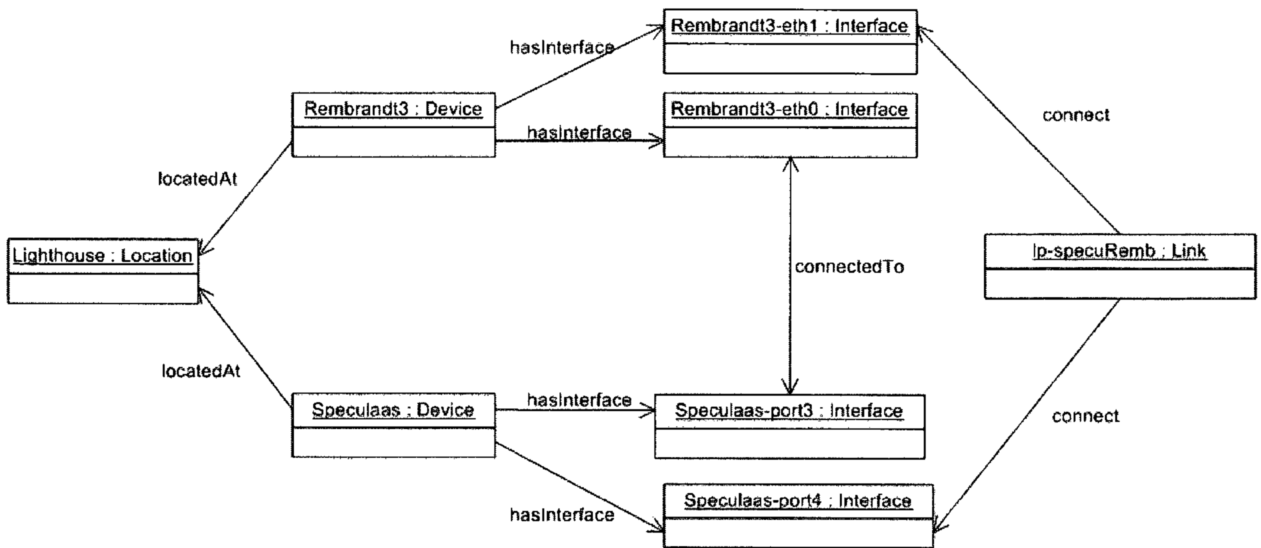

Figure 16: UML instance diagram of a NDF description of a simple network

In Figure 16, the UML instance diagram shows that the network has two devices, Rembrandt3 and Speculaas, which are located at "Lighthouse". The interface Rembrandt3-eth0 of device Rembrabdt3 is directly connected to the interface Seculaas-port3 of device Speculaas. A link called lp-specuRemb connects two other interfaces of the two devices.

Since the NDL files are based on RDF, RDF query languages such as SPARQL can be used to get information out of the NDL files. The following shows an example of a SPARQL query using the example in Figure 16.

```
SELECT ?device1 ?device2
WHERE { ?if1  ndl:connectedTo ?if2 .
        ?if2  ndl:connectedTo ?if1 .
        ?device1 ndl:hasInterface ?if1.
        ?device2 ndl:hasInterface ?if2  }
```

The example shows that the query selects two devices, represented by the variables ?device1 and ?device2, which must meet the constraints specified in the where clause. This clause uses two additional variables ?if1 and ?if2 which must be connected to each other and must also belong to the devices. This query will return all device pairs that are directly connected to each other. When this query is applied to a directory that

contains the network shown in Figure 16, the two devices of the network will be returned as result. The result will be presented as a string containing the two device name separated by a space, in the form "Rambrandt3 Speculaas".

The main purpose of NDL is to describe network topologies. By exchanging the NDL documents among collaborating GLIF networks, various provisioning tools such as UCLP, DRAC, and DRAGON, can understand each other's network topology. Finding a suitable path across multiple domains using the topology information contained in the NDL files greatly benefits the resource discovery process across multi-domains for end-to-end connection provisioning. It is also beneficial for trouble-shooting along the lightpath in case faults. Currently, NDL is still under development. The NDL schemas are evolving as new vocabularies are added. Several GLIF organizations, such as SURFnet and Canarie, have used NDL to describe their network topology. More and more GLIF organizations are following SURFnet and Canarie to create NDL files for their networks.

5.3.5. Basic design choices for UCLP resource adverting and discovery

Following the system requirements described in Section 5.2 and the considerations explained above, we see that the UDDI registry and the Jini lookup registry are not suitable for resource advertising and discovery for UCLP system. Neither the UDDI's keyword-matching lookup nor the Jini lookup registry's template-matching process satisfies the basic system requirements. The data model of the UDDI registry cannot describe the relations among the UCLP resources. The lookup process of the UDDI registry is limited to the keyword-matching which uses the key-value pairs of values as strings. The UDDI lookup process cannot give an answer for a range searching. Although The Jini lookup registry can satisfy the basic system requirement (a) which is looking for resources by explicit types, it still cannot fulfill the requirement of range searching which is specified in the system requirement (b). The RDF data model not only can describe the UCLP resources lists and the relationship between resources contained in a resource list very well, it also provides a directory which is called Sesame with query support that satisfies the basic system

requirements (a) and (b) which are explicit lookup as well as range searching respectively. Furthermore, by adopting the NDL topology schema in the RDF schema for the UCLPv2 resource lists, the capability of sharing resources with other applications is achieved. The RDF/Sesame database has been chosen as the directory for our resource adverting and discovery mechanism. In order to fulfill the other design requirements, we propose a system of resource advertising and discovery based on the RDF/Sesame directory as described in the next sections.

5.4. Design of resource advertising and discovery for UCLP

In this section, we elaborate the design of resource advertising and discovery for the UCLPv2 system based on the system requirements and basic design choices described in Sections 5.2 and 5.3.

5.4.1. System architecture

We propose a system architecture which integrates the new functions of resource adverting, discovery and leasing with the existing UCLPv2 system. The new system functions are realized by integrating the client GUI, the Sesame directory, and the Web Service of resource list downloading modules into the current UCLPv2 system. Figure 17 shows here two such systems form an architecture for resource advertising and discovery.

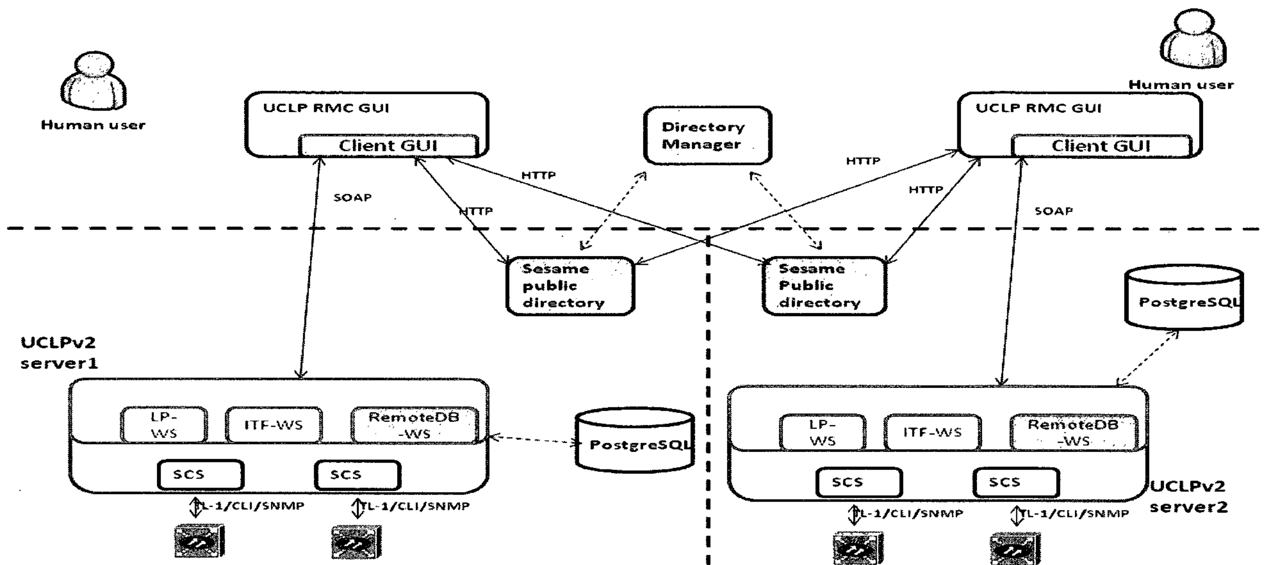

Figure 17: System architecture

The system modules for resource adverting and discovery are shown in Figure 17 with dark color background. They are the following:

- Client GUI: The graphical user interface which provides functions for UCLPv2 organizations to advertise and discover resources.
- Sesame public directory: The directory for storing, searching the advertised resource lists.
- Directory manager: A global Sesame directory that collects all the public Sesame directory addresses. Its address is well-known to these UCLPv2 public Sesame directories.
- RemoteDB-WS: The Web Service that provides for uploading/downloading of resource lists to/from the UCLPv2 server database.

5.4.2. Resource list transformation

A resource list, which is an XML file, must be transformed into RDF format to be stored in the Sesame directory. The transformed resource list is called RDF-RL. There is lot of information in a resource list which is used for UCLPv2 GUI display purpose. For example, a virtual node has locationX, locationY, width and height attributes which are used by the GUI to display the shape and the position of the node. This information should not be

transformed into the RDF-RL. However, we would like to transform the resources that are contained in the resource list and the relations between resources as well as the network topology information into the RDF-RL.

According to the data structure of the resource list depicted in Figure 13 and the NDL topology data structure shown in Figure 15, a UML diagram of a RDF-RL is obtained and shown in Figure 18.

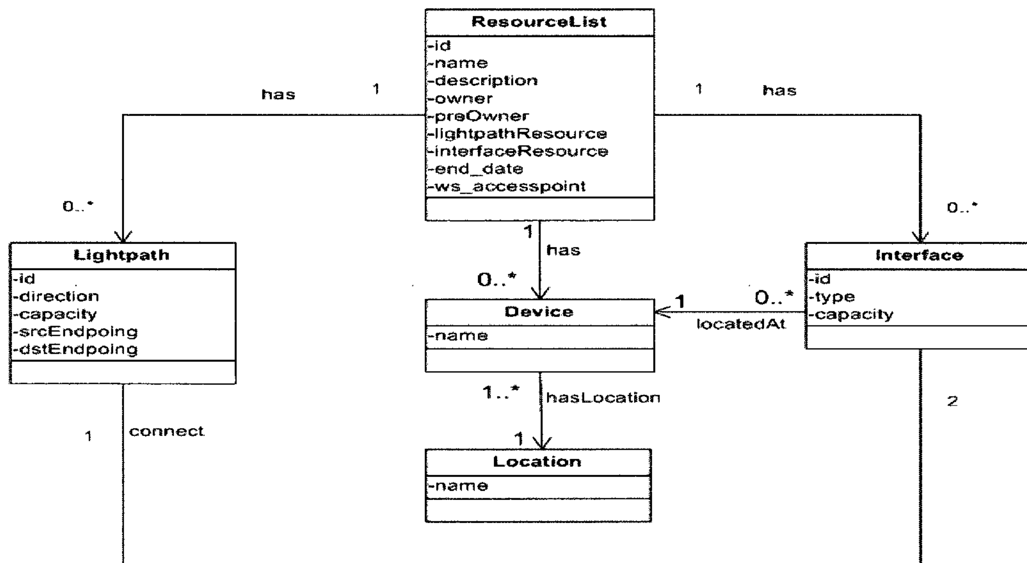

Figure 18: An UML diagram of RDF-RL data structure

The diagram in Figure 18 illustrates that a resource list has devices, lightpaths and interfaces. A lightpath connects two interfaces and one device has many interfaces. A location also has many devices. The individual resources (lightpaths and interfaces) are derived from the UCLPv2 resource list data model shown in Figure 13. The relationships among resources as well as the topology information are derived from the NDL topology schema as shown in Figure 15.

The Resourcelist, the Lightpath and the Interface of the RDF-RL are derived from the APNContainer, the Lightpath WS-Resource and the Interface WS-Resource of the UCLP resource list respectively (see Figure 13). The Device and location of the RDF-RL

correspondent to the network element and location information that contained in the VirtualNode of the UCLP resource list (see Figure 13). All the attributes of the RDF-RL are also derived from the correspondent attributes of the UCLP resource list.

Comparing the RDF resource list structure in Figure 18 with the NDL topology schema data structure describes in Figure 15, we can see that the link of the NDL topology schema are now called the lightpath in the RDF resource list. The link and the lightpath both have the same meaning, namely a network link. Except the ResourceList class and its "has" relationships, all the relationships among the Lightpath, the Interface, the Device and the Location described in Figure 18 are following the relationships that are defined in the NDL topology schema.

Following the correspondence explained above, a RDF-RL is obtaind from its corresponding resource list (RL) as follows:

- The RDF ResourceList is transformed from the APNContainer of the RL. The attributes of the ResourceList such as id, name, owner, perowner correspondent to the APNcontainer arrtibutes such as id, name, owner, oOwner, respectively.
- The RDF LightpathResource is transformed from the lightpath resource of the RL. The attributes of the LightpathResource such as id, direction and capacity are the same as the id, direction and capacity of the lightpath resource attribute.
- The RDF InterfaceResource correspondents to the interface resource of the RL. All it's attributes are transformed from the attributes of the interface resource.
- The RDF Device and Location are both transformed from the RL VirtualNode. The VritualNode attribute "name" is the "name" attribute of the Device. The Location is derived from the attribute "location" of the VirtualNode.

This transformation keeps the network topology information in the resulting RDF-RL. Since UCLPv2 cannot process RDF data, an organization will store the original resource list XML file in the PostgreSQL database table as a binary object. A Web Service called "RemoteDB-WS" provides downloading operation for transferring the XML file to a user's local UCLPv2 RMC client GUI workspace when the corresponding RDF-RL has been discovered. The attribute "ws_accesspoint" of the ResourceList represents the URL of the RemoteDB-WS Web Service. The RemoteDB-WS is explained in Section 5.4.7.

5.4.2.1. RDF schema for UCLPv2 RL

A RDF schema is defined according to the data model of the RDF-RL which is illustrated in the UML diagram in Figure 18. The schema is defined as follows: The classes in the UML diagram correspond to the RDF classes defined in the Schema, whereas the attributes of the UML classes are defined as the properties in the RDF schema. The associations between RDF properties and classes are defined in the schema as well. In order to be compatible with NDL, the schema uses as far as possible the definitions defined by NDL schema. If there is no corresponding definition in NDL schema, then a new definition is introduced. The following RDF classes are defined:

- **ResourceList**: the resource list class. This is the newly defined class, not included in NDL.
- **Interface**: represents the interface resources in a RL. This definition comes from NDL.
- **Lightpath**: represents the lightpath resource in a RL. This is a newly defined class.
- **Device**: represents a network element. This definition comes from NDL.
- **Location:** represents a place of a collection of network elements. This definition comes from NDL.

The following RDF properties have been defined:

- ID: the identifier of a resource. This property is applied to all resources and it is not from NDL.
- name: the name of a resource. It is applied to the classes Device, Location and ResourceList. It is defined in NDL.
- owner: defines the current owner of a RL. It applies to the class ResourceList. It is a new definition.
- pre-owner: the previous owner of a RL. It applies to the class ResourceList as well. It is also a new definition.
- hasDevice: this property assigns devices to a RL. It is a new definition.
- hasInterface: this property assigns interface resources to a RL which is defined by NDL.
- hasLightpath: this property assign lightpath resources to a RL which is newly defined.
- endDate: indicates the lifetime of a RL in the directory. It is a newly defined property.
- wsAccessURL: the Web Service access point address for downloading the XML format RL from the database Web Service.
- capacity: the bandwidth of a lightpath or an interface, lightpath resource. It is a float number.
- description: human-readable description for the subject. It is applied to all resources

- locatedAt: The relation between a device and an interface is defined by the locatedAt property. It is applied to interface class.
- hasLocation: this property assigns a location to a device.
- srcInterface: this property assigns a interface as source location to a lightpath.
- dstInterface: this property assigns a interface as destination location to a lightpath.
- direction: indicates a lightpath is unidirectional or bidirectional. The value of this property is "2way" or 1way".
- type: this property defines an interface type based on the interface technology.
- isLeased: this property is applied to class ResourceList. It indicated whether a RL is leased or not. This is newly defined property.

An example of defining classes and properties in the RDF-RL schema is shown in the following. The entire schema can be found in Appendix A.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:ndl= "http://www.science.uva.nl/research/sne/ndl#" >
    <rdfs:Class rdf:ID="ndl:Location">
        <rdfs:label xml:lang="en">Location</rdfs:label>
        <rdfs:comment xml:lang="en">The location of a network device.
    </rdfs:comment>
    </rdfs:Class>
    <rdfs:Class rdf:ID=" ndl:Device">
        <rdfs:label xml:lang="en">Device</rdfs:label>
        <rdfs:comment xml:lang="en"> network device. </rdfs:comment>
    </rdfs:Class>
    <rdfs:Class rdf:ID="Lightpath">
        <rdfs:label xml:lang="en">Lightpath</rdfs:label>
    </rdfs:Class>
    <rdf:Property rdf:ID="srcDevice">
        <rdfs:label xml:lang="en">source device</rdfs:label>
        <rdfs:comment>The relation between a Device and a lightpath. The device is
    a source device of a lightpath. </rdfs:comment>
        <rdfs:domain rdf:ID="Lightpath" />
        <rdfs:range rdf:ID="ndl:Location" />
```

```
        </rdf:Property>
        <rdf:Property rdf:ID=" ndl:name">
                <rdf:type
        rdf:resource="http://www.w3.org/2002/07/owl#DeprecatedProperty" />
                <rdfs:label xml:lang="en">name</rdfs:label>
                <rdfs:comment>A short human-readable name for the
        subject.</rdfs:comment>
                <rdfs:domain rdf:ID="rdfs :Resource" />
                <rdfs:range rdf:ID="rdfs:Literal" />
        </rdf:Property>
</rdf:RDF>
```

An example of a RDF-RL which has a lightpath and two interface resources is shown as the following:

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3c.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:rsLst="http://uclp02.canarie.ca/schema/rdf/resourcelist#"
                xml:base="http://uclp02.canarie.ca/uclp/term" >

<rsLst:ResourceList  rdf:ID="ResourceListCANARIENetworkHPDMnetRL1">
        <rsLst:name> CANARIENetwork-HPDMnet RL1</rsLst:name>
        <rsLst:owner> HPDMNET</rsLst:owner>
        <rsLst:preOwner>CANARIE</rsLst:preOwner>
        <rsLst:endDate                rdf:datatype="&xsd;                dateTime">2009-10-
21T20:00:00</rsLst:endDate>
<rsLst:description> The resource list is reserved for HPDNNet </rsLst:description>
<rsLst:wsAccessURL
rdf:resource="https://uclp02.canarie.ca:8080/wsrf/services/core/DBService" />
<rsLst:hasInterface rdf:resource="#FedeGigaPoP" />
<rsLst:hasInterface rdf:resource ="#CRC-eth0" />
<rsLst:hasInterface rdf:resource ="#STS12-Tor-Ott" />
<rsLst:hasInterface rdf:resource ="#STS12-Ott-Tor" />
<rsLst:hasLightpath rdf:resource ="#CANARIE-Toronto-Ottawa-031239" />
        <rsLst:hasDevive rdf:resource="# TORO1OME1" />
        <rsLst:hasDevice rdf:resource="#OTWA1OME1"/>
</rsLst:ResourceList>
<rsLst:Location rdf:ID="Tornoto">
```

```
<rsLst:name>Toronto</rsLst:name>
</rsLst:Location>
<rsLst:Location rdf:ID="Ottawa">
        <rsLst:name>Ottawa</rsLst:name>
</rsLst:Location>
<rsLst:Device rdf:ID="TORO1OME1">
        <rsLst:name>TORO1OME1</rsLst:name>
        <rsLst:locateAt rdf:resource="#Toronto" />
</rsLst:Device>
<rsLst:Device rdf:ID="OTWA1OME1">
        <rsLst:name>OTWA1OME1</rsLst:name>
        <rsLst:locateAt rdf:resource="#Ottawa" />
</rsLst:Device>
<rsLst:Interface rdf:ID="FedeGigaPoP">
        <rsLst:type>Ethernet</rsLst:type>
<rsLst:capacity>1000</rsLst:capacity>
<rsLst:unit rdf:datatype="&xsd;string">Mbps</rsLst:unit>
<rsLst:locateAt rdf:resource="#TORO1OME1" />
</rsLst:Interface>
<rsLst:Interface rdf:ID="CRC-eth0">
        <rsLst:type>Ethernet</rsLst:type>
<rsLst:capacity rdf:datatype="&xsd;float">1000</rsLst:capacity>
<rsLst:unit rdf:datatype="&xsd;string">Mbps</rsLst:unit>
<rsLst:locateAt rdf:resource="#OTWA1OME1" />
</rsLst:Interface>
<rsLst:Interface rdf:ID="STS12-Toro-Ott ">
        <rsLst:type>SONET</rsLst:type>
<rsLst:capacity rdf:datatype="&xsd;float">622.08</rsLst:capacity>
<rsLst:unit rdf:datatype="&xsd;string">Mbps</rsLst:unit>
<rsLst:locateAt rdf:resource="#TORO1OME1" />
</rsLst:Interface>
<rsLst:Interface rdf:ID="STS12-Ott-Tor">
        <rsLst:type>Ethernet</rsLst:type>
<rsLst:capacity rdf:datatype="&xsd;float">1000</rsLst:capacity>
```

```
<rsLst:unit rdf:datatype="&xsd;string">Mbps</rsLst:unit>
<rsLst:locateAt rdf:resource="#OTWA1OME1" />
</rsLst:Interface>
<rsLst:Lightpath rdf:ID="CANARIE-Toronto-Ottawa-031239">
        <rsLst:direction>2way</rsLst:direction>
<rsLst:capacity rdf:datatype="&xsd;float">622.08</rsLst:capacity>
<rsLst:unit rdf:datatype="&xsd;string">Mbps</rsLst:unit>
<rsLst:srcInterface rdf:resource="#STS12-Tor-Ott " />
<rsLst:dstInterface rdf:resource="#STS12-Ott-Tor" />
</rsLst:Lightpath>

</rdf:RDF>
```


5.4.3. Sesame public directory

The Sesame will be used as the directory for storing, searching the advertised RDF-RLs. Since organizations in the UCLPv2 system are administratively independent entities and each organization defines an ID and a name to identify their RLs within the organization, it is possible to use one public directory for resource advertising and discovery for all the UCLPv2 organizations. Clients can access the Sesame server via the HTTP protocol and through a local Java API.

In theory, one public Sesame directory should meet the requirements for resource adverting and discovery for all parties that use UCLP systems. However, due to the administrative responsibility, security and firewall issues, we propose that each UCLPv2 system includes a Sesame directory. In order to let the directories of different UCLPv2 systems know each other, we propose the concept of a directory manager. The directory manager is also a Sesame directory with a well- known address. The purpose of the directory manager is to collect all the public directories addresses. Each public directory has the option to register itself to the directory manager and query the directory manager for other public Directories. Or it can choose to be a local directory servicing only its local UCLPv2 system instance.

For the purpose of inter-working with other systems, we propose that other systems, such as Chronos on DRAC, discover the advertised resource lists of the UCLPv2 systems that have advertised in the public directories. The access to a public Sesame directory from other systems is via user name/password authentication.

5.4.4. Naming convention for RLs

UCLPv2 system uses the name of a RL as the identification of that RL. In order to identify resource lists that are advertised by different organizations, the name of a RL must be unique. Current systems let admin users define a name for a RL as they want. This may cause duplicated naming. One solution is to this problem is to use the organization's name as the prefix for the RL name. For example, the organization of Canarie may export a resource list with a name of "CANARIE.HPDMNet.001". Here the string HPDMNet may indicate that this RL is for a project called HPDMNet.

5.4.5. Advertising, discovery and leasing policy

In our resource advertising and discovery system, we introduce the concept of resource leasing. The idea of resource leasing is to transfer the ownership of the resource list to the leasing organization. An organization gains control of the resources in an advertised resource list by becoming the owner of that advertised resource list through a leasing operation. Each resource list has an end date which indicates the resource list's lifetime of being advertised. When it reaches the end date, the resource list's advertisement expires.

The UCLP user that advertises and discovers resources must follow the following policy procedures:

- Resource lists can be advertised publicly by exporting the resource list to an organization called "Public".
- Resource lists can be reserved for a given organization. Other organizations will not be allowed to discover and lease such resource lists.

- Every advertised resource list must have an end date to indicate its lifetime in the directory. The end date can be extended by the administrator of the advertising organization.
- Users can discover and lease the resource lists that are advertised publically and, or those that reserved for them.
- Users must specify a lease end date when leasing a resource list. The end date must not be after the resource list's end date.
- Users can release a leased resource list before its leasing is expired. Upon the lease end date, a system function should be able to release the resource list automatically. If there is any connection created using the leased resources, the connection will be deleted and the resources will be released.
- Users can cancel an advertised resource list which is advertised by the users only if the resource list is not leased. A system function is able to cancel the advertisement automatically upon reaching the resource list's end date. The resource list will then be deleted from the directory.

5.4.6. A typical use of resource list advertising and discovery

Before describing the implementation of the resource adverting and discovery system in Chapter 6, we would like to explain the process of advertising and discovery via a use case. The purpose is to understand the process of resource advertising, discovering and leasing for designing the corresponding functions. In this example, we consider three organizations, called Canarie, Environment Canada (EC) and CRC which participate in resources sharing. EC and CRC will import resource lists. Canarie has created and exported three resource lists called "CANARIE-EC-001", "CANARIE-CRC-001" and "CANARIE-public-001". The RL "CANARIE-EC-001" is exported to EC by changing the owner to EC; the RL "CANARIE-CRC-001" is exported to "CRC" while the last one "CANARIE-public-001" is exported to "public". After the admin user of Canarie selected the resource lists one by one, the three RLs have been transformed into RDF-RLs using the XML-RDL transformation function build in the client GUI. Then the three RDF-RLs are uploaded to the public directory. And at the same time, the three XML resource lists are also uploaded to the UCLPv2 server's database via the RemoteDB_WS Web Service. The procedure for advertising a resource list is illustrated by the UML sequence diagram in Figure 19

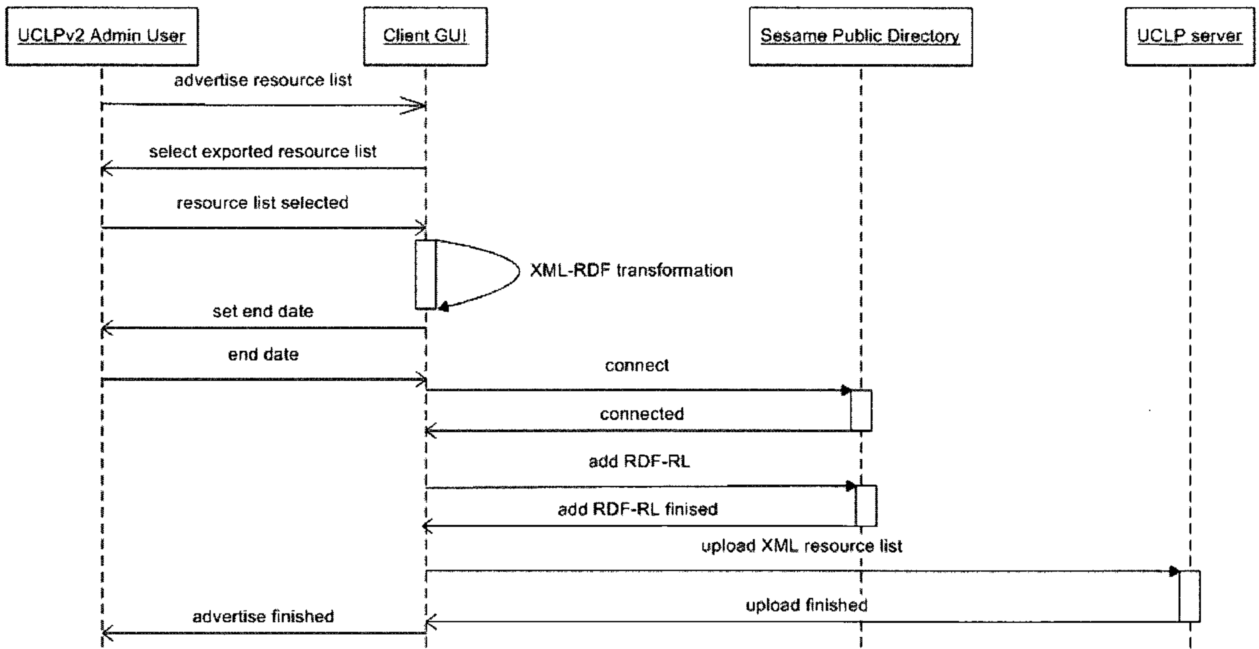

Figure 19: UML sequence diagram for advertising a resource list

Next, the EC and CRC admin users, independently, launch their UCLPv2 RMC client GUI to discover the RDF-RLs in the public directory. Both EC and CRC admin users execute the function "discover by reservation" to look for the resource list that were reserved for them. The EC admin user will see a RDF-RL with the name "CANAREI-EC-001" displayed in the GUI whereas the CRC admin user will see that the "CANARIE-CRC-001" is reserved for CRC. Then the EC and CRC admin users decide to lease the reserved resource lists. A leasing end date will be specified during the leasing process and the original XML resource list will be downloaded into the local workspace. The value of the property "isLeased" is changed from "false" to "true" for both of the reserved RDF-RL in the directory. The UML sequence diagram of Figure 20 shows the procedure of resource discovery.

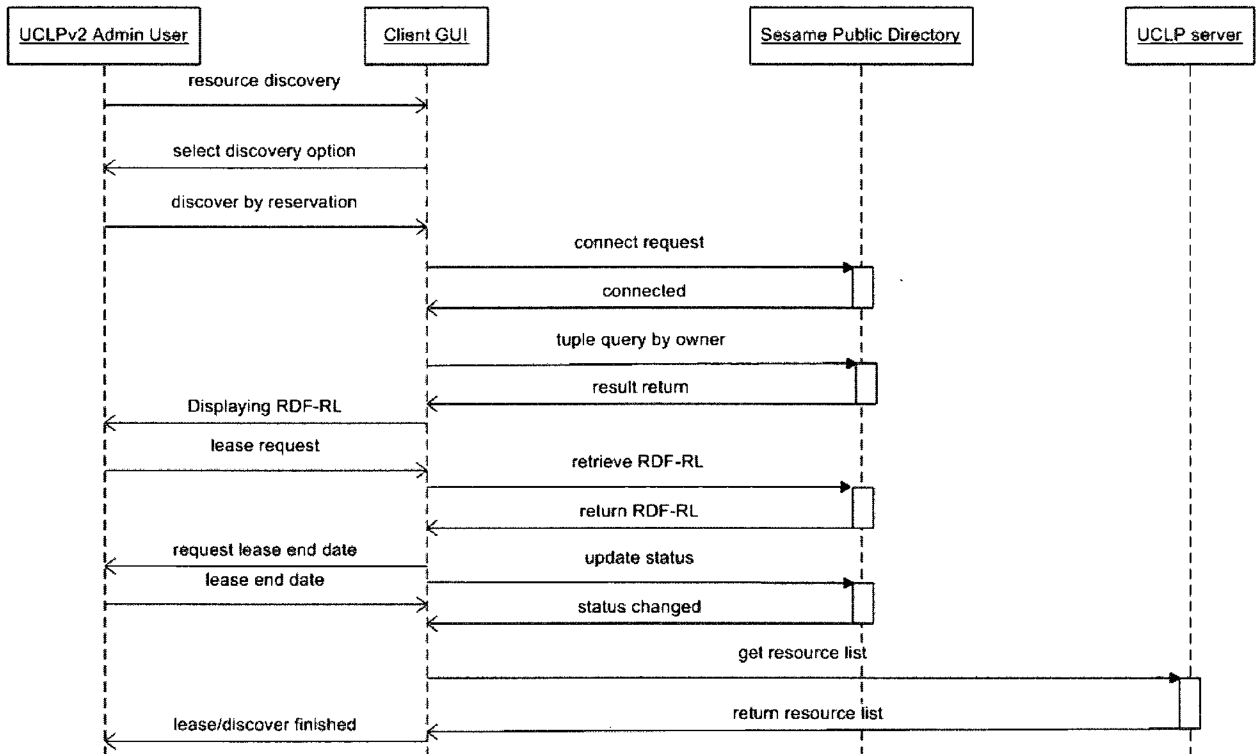

Figure 20: Sequence diagram for discovering a reserved resource list

In order to look for the publicly advertised resource "CANARIE-public-001", either the admin user of EC and CRC can use the browse option to look into the public directory. Both the admin users can see the "CANARIE-public-001" via their client GUI. We assume that the EC admin user will lease the resource list. So the EC admin user selects the RDF-RL "CANARIE-public-001" and leases it. The procedure of discovering publicly advertised RDF-RL is the same as the one shown in Figure 20.

5.4.7. Functions for resource advertising, discovery and leasing

The resource advertising and discovery functionalities in the client system enable UCLPv2 users to register/deregister, discover RDF-RLs in the directory. Figure 21 shows the system components which are integrated into the existing UCLPv2 system.

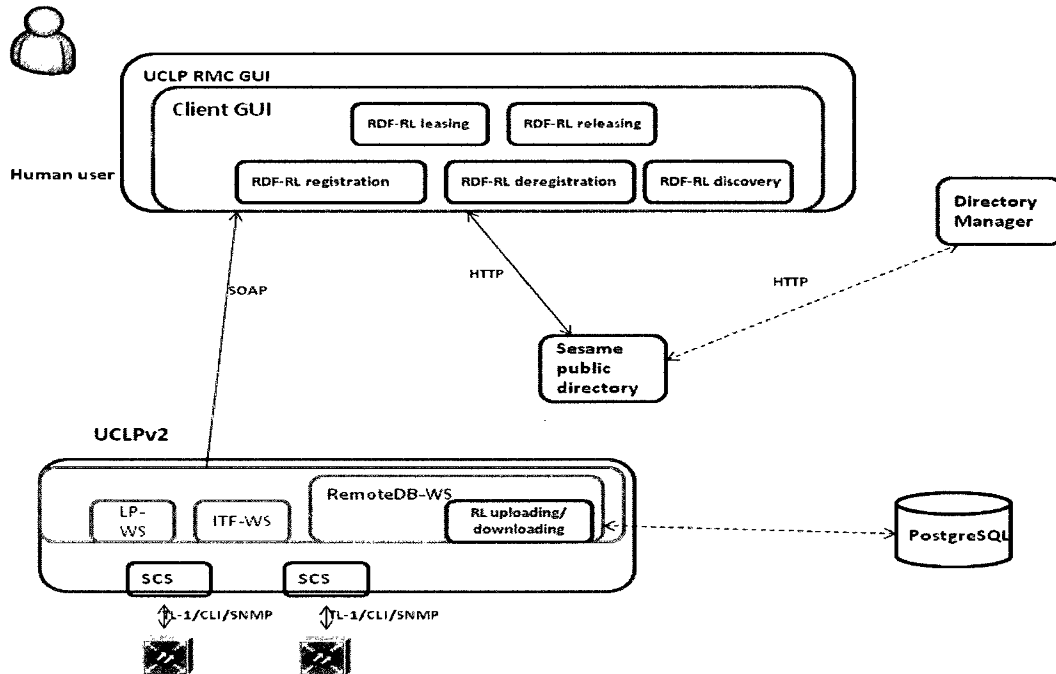

Figure 21: System components of resource adverting and discovery

The system components are the client GUI, the Sesame directory, the directory manager and the RemoteDB-WS. The functions, contained in the client GUI, use the Sesame APIs to communicate with the directory for resource advertising and discovery. The following is a description of these functions:

- **RDF-RL registration**: The main purpose of this function is to upload the RDF-RLs to the directory and to set up the end date for the advertised RDF-RL. It has two steps: resource list transformation and resource list uploading. The resource transformation converts an XML resource list into a RDF-RL and configures the end date for the RDF-RL. The resource list uploading involves a connection to the Sesame directory and adding the RDF-RL to its database.
- **RDF-RL deregistration**: This function will delete a RDF-RL in the directory. This is also used for canceling an advertisement when the RDF-RL reaches its end date. All the resources that are contained in the RDL-RF are removed from the directory. The process of deregistration must comply with the policies about resource adverting defined in Session 5.4.5.
- **RDF-RL discovery**: This function lets the UCLPv2 admin users discover RDF-RLs and their contents in the directory using the browse method or query method. The browse method displays all publicly advertised RDF-RLs and the query method returns a result based on the specified queries. It should support the SPARQL, SeRQL and RQL query languages.

To fulfill the design requirements of resource leasing, the following functions are presented:

- **RDF-RL leasing**: This operation allows users to lease a RDF-RL including all the resources in that RDF-RL. This operation will set up the lease end time. The process involves of downloading the desired resource list from the UCLPv2 server database via the function "RL downloading" as showed in the diagram of Figure 21. The RDF-RL in the directory will be marked as "leased" to indicate that the RDF-RL has been leased. The owner of the RDF-RL is changed to the leasing organization.
- **RDF-RL releasing**: The purpose of this function is to release a leased RDF-RL. It involves updating the status of the RDF-RL from "leased" to "available" in the directory. If the RDF-RL is publicly advertised, then the owner will be changed back to public.

The functions below are support functions that allow UCLPv2 admin users to upload the original resource list to the UCLP server database or download the original resource list to their local workspace. They are implemented as Web Service functions via RemoteDB-WS. The existing UCLPv2 system does not have functions to use RDF data directory in the creation of end-to-end connections. Even if an admin user discovers and leases the desired RDF-RL in the public Sesame directory, there is still a need to pass the original XML format resource list to the corresponding organization. One implementation decision has been made which is to build a Web Service to provide a download operation for admin users or other applications to get the resource list from the UCLPv2 server database. After a resource list has been transformed into a RDF-RL and advertised in the public directory, the original resource list will be uploaded to the UCLP server database. The uploaded resource list can be downloaded by calling the downloading function provided by the Web Service.

- **RL uploading**: After the RDF-RL is advertised in the Sesame directory, the original resource list will be uploaded to the UCLP server database via the Web Service.
- **RL downloading**: When users find a desired RDF-RL, then the RL will be downloaded via this Web Service from the UCLP server database by using the name of the resource list which is obtained from the name of the RDF-RL.

Chapter 6 - Detailed system design and Implementation

The proposed system of resource advertising and discovery consists of a Sesame directory and a client GUI which integrates functionalities for resource advertising and discovery as shown in Figure 17 of Chapter 5. This chapter briefly describes the Sesame directory implementation and focuses on the implementation of the client GUI and its functionalities. Furthermore, an example usage of the resource advertising and discovery is described which is used to validate the implementation of the system.

6.1. Sesame directory implementation

Sesame is installed and configured to run in server mode. Client applications access Sesame's repositories via the HTTP protocol or locally through Sesame APIs. One repository is created which has the name "UCLPPublic" suffixed by the host name of the UCLP server to which it belongs. The procedures for installing and configuring a Sesame server are described in [23]. A tool called Sesame console is used to configure the Sesame server.

6.1.1. Implementing the directory manager

As mentioned in section 5.4.3, a directory manager is a Sesame repository which has the name "DirectoryManager" on a Sesame server. The directory manager does not belong to any UCLPv2 server and its address is well-known. The purpose of the directory manager is to register the addresses of all the public directories. Consequently, it allows UCLPv2 administrators to query other public directories for resource discovery. Every public UCLPv2 directory has the option to register itself with the directory manager at its boot-up time if such a directory manager exists. The registration involves uploading some RDF data to the directory manager. The RDF data contains RDF descriptions of the public directory. The

following is an example of a RDF file containing the registration data for a public directory called "UCLPPublic.uclp02".

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:sys="http://www.openrdf.org/config/repository#">
        <rdf:Description
        rdf:about="http://uclp02.canarie.ca/sesame/UCLPPublic.uclp02">
                <rdf:type
                rdf:resource="http://www.openrdf.org/config/repository#Repository"/
                >
                <sys:repositoryID>UCLPPublic.uclp02</sys:repositoryID>
        </rdf:Description>
</rdf:RDF>
```

When the administrator of an organization wants to discover resources from another UCLP server, he would use a client GUI program which is integrated into the UCLPv2 GUI to select the desired public directory of that server for resource discovery.

6.2. Client GUI program

The client GUI provides a graphical user interface for organization administrators to advertise and discover resources. The client GUI is implemented as an Eclipse Rich Client Platform (RCP) wizard dialogue and integrated into the UCLPv2 GUI. It provides three main functions: Resource advertising, Resource discovery and Resource leasing. A cancel operation is provided to cancel advertisements or leases. The resource advertising procedure is described by an UML activity diagram shown in Figure 22. A resource list is required to be exported before it can be advertised. The exportation is done via the UCLPv2 GUI function "Export". Then the client GUI allows an admin user to select the exported resource lists and advertise them in the public directory. Before the selected resource list is advertised, it will be transformed from XML to RDF. The end date of the resource list is also specified. If the connection to the Sesame directory fails, the procedure will be terminated after a preconfigured timeout period. The resource discovery function provides three options: (1) Browse the directory; (2) discover reservations which returns those RDF-RL reserved for that organization; (3) discover by query in the public

directory. The lease function will lease the desired RDF-RL. When the admin user selects a desired resource list, the client GUI invokes the resource downloading Web service to download the resource list.

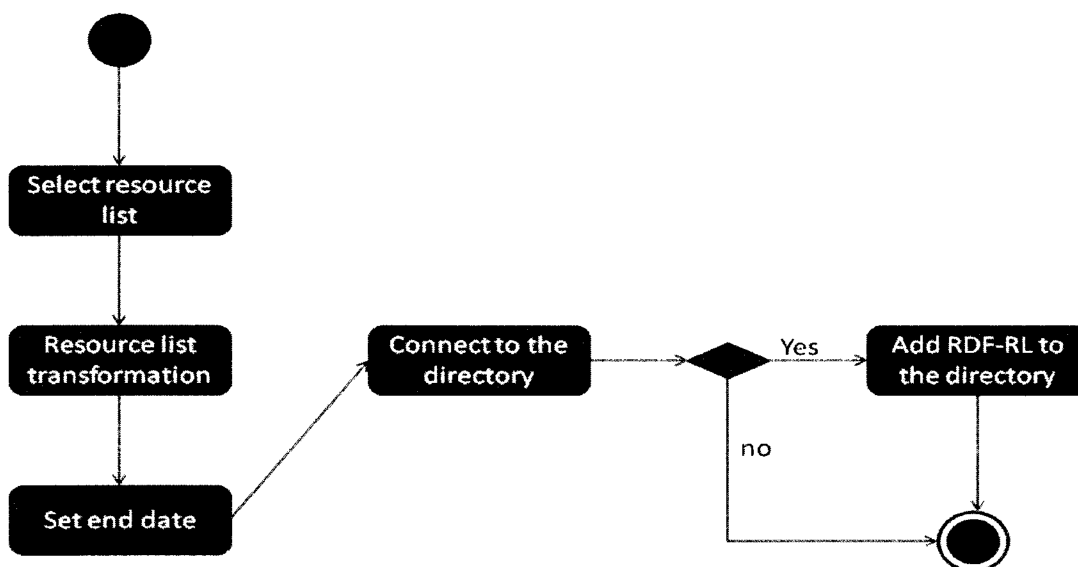

Figure 22: UML activity diagram for the procedures of resource advertising

6.3. XML to RDF

One important function that is implemented in the client GUI is the function which converts an XML-formatted resource list into one in RDF format. We have implemented this function using XML transformations defined by an XSL stylesheet. The function uses the Java API for XML Processing (JAXP) version 1.1 which supports SAX 2.0, DOM level 2, and XSL transformations. The JAXP provides an XSL transformation engine, which takes an XML document as input and uses the XSL transformation stylesheet to create a new document format as shown in Figure 23. Depending on the definitions in the XSL stylesheet, the resulting document can be an XML document with different structures.

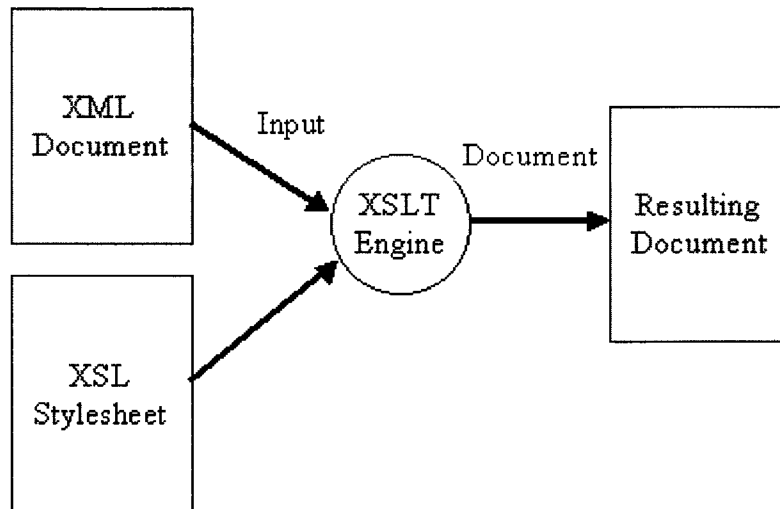

Figure 23: Using an XSLT engine [27]

In order to convert an XML RL into a RDF-RL, we designed an XSL stylesheet that defines this translation. The following is a part of this stylesheet showing a template rule definition for converting the root element <APNContainer> of a RL into a RDF statement of the form of <ResourceList>.

```
<xsl:stylesheet version="1.0"
xmlns:xsl=http://www.w3.org/1999/XSL/Transform
xmlns="http://www.w3.org/TR/xhtml1/strict"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rsLst="http://uclp02.canarie.ca/schema/rdf/resourcelist#">
<!—specify the output format is XML  -- >
<xsl:output method="xml" omit-xml-declaration="no"/>
<xsl:template match="/">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rsLst="http://uclp02.canarie.ca/schema/rdf/resourcelist#"
         xml:base="http://uclp02.canarie.ca/uclp/term">
        <xsl:apply-templates/>
</rdf:RDF>
</xsl:template>
<xsl:template match="APNContainer">
<xsl:element name="rsLst:ResourceList">
<xsl:attribute name="rdf:ID">
<xsl:value-of select="@name">
</xsl:attribute>
</xsl:element>
<rsLst:name><xsl:value-of select="@name"></rsLst:name>
```

```
<rsLst:owner><xsl:value-of select="@owner"></rsLst:name>
<rsLst:preOwner><xsl:value-of select="@oOwner"></rsLst:preOwner>
<xsl:element name="rsLst:endDate">
                <xsl:arrtibute name="rdf:dataType">
                <xsl:vaule-of select="@endtime">
                </xsl:arrtibute>
</sxl:element>
</rsLst:ResiurceList>
</xsl:template>
</xsl:stylesheet>
```

The example shows that the first an element <APNContainer> is transformed into <rsLst:ResourceList> with an attribute named "rdf:ID" obtains as value the name of the APNContainer. Next the attributes of the element <APNContainer> are converted into elements <rsLst:name>, <rsLst:owner> and <rsLst:prewoner> using the attribute values as element content. Therefore an APNContainer element defined as follows:

```
<APNContainer mname="Resource Lists/CANARIE HPDMnet RL 1.jpg"
        exported="true" ln="false" oOwn="CANARIE" own="HPDMnet"
st="This is a Resource List" puri="N/A" sll="true" name="CANARIE HPDMnet RL 1"
endtime="2009-10-21 20:00:00" >
```

would be transformed into the following RDF structure:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:rsLst="http://uclp02.canarie.ca/schema/rdf/resourcelist#"
                xml:base="http://uclp02.canarie.ca/uclp/term" >
<rsLst:ResourceList  rdf:ID=" CANARIE HPDMnet RL 1">
                <rsLst:name> CANARIE HPDMnet RL 1</rsLst:name>
        <rsLst:owner> HPDMNET</rsLst:owner>
        <rsLst:preOwner>CANARIE</rsLst:preOwner>
        <rsLst:endDate rdf:datatype="&xsd; dateTime">2009-10-
                21T20:00:00</rsLst:endDate>
        </rsLst:ResiurceList>
</rdf:RDF>
```

An implementation of the XSL translation engine is obtained by using the java.xml.transform package provided by SUN. The transformation of an XML file into its

corresponding RTF document is provided by the convert method defined in the following Java class. Note that the file "resourceList.xsl" contains the XSL definition of the desired transformation.

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.io.*;

public class xml2rdf {
  /* constructor */

      public xml2rdf(){}

/* xml2rdf convert method. It takes one XML file as input and convert the
* file to RDF-RL using the XML stylesheet named resourceList.xsl
* The first input string variable is the XML file, or the RL, the second
* input variable is the name of the output file, which is the RDF-RL
*/

public void convert(String xml_file, String to_file)
    throws TransformerException,
    TransformerConfigurationException,
           FileNotFoundException, IOException
  {
        TransformerFactory tFactory = TransformerFactory.newInstance();
        Transformer transformer = tFactory.newTransformer(
                  new StreamSource("resourceList.xsl"));
        transformer.transform(new StreamSource(xml_file),
            new StreamResult(new FileOutputStream(to_file)));

      return;
  }
}
```

6.4. The RL downloading Web Service

The downloading Web Service which is called "RemoteDB_WS" has an interface which provides operations to retrieve the XML format RL from the UCLPv2 server database. UCLPv2 servers use a relational database PostgreSQL to persistently store the WS-Resources such as endpoint WS-Resources which have been used by the resource virtualization layer to form lightpath resources and interface resources. A table which has a name "remoteDB_reource" is created to store the XML resource list as an Object Identifier

(OID) in the table. A part of the WSDL file of the downloading Web Service is illustrated in Figure 24. Since the WSDL is an XML file, an UML diagram is used to present the definition of the WSDL. The WSDL of the RemoteDB_WS can be found in Appendix C.

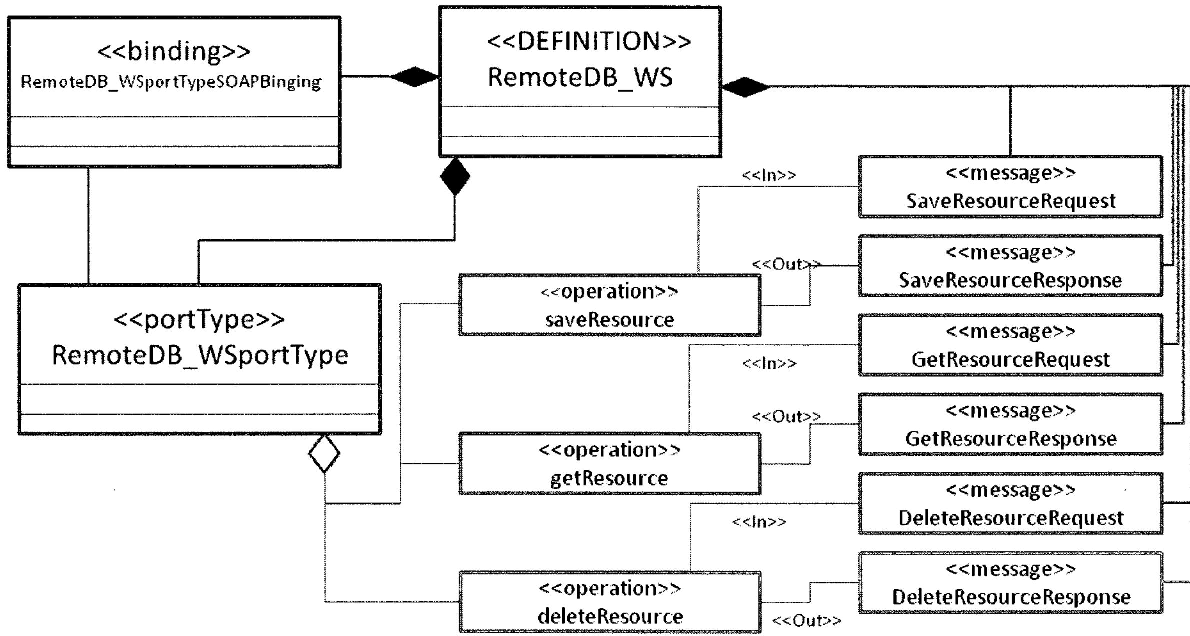

Figure 24: UML presentation of Web Service "RemoteDB_WS"

The WSDL file of RemoteDB_WS has defined one Interface as the RemoteDB_WSportType which provides three oprations: saveResource, getResource and deleteResource. Each operation has in/out messages defined.

A Java class called "RemoteDBWSWrapper" has been defined to call the Web Service. This class provides a single point to call all the remote database Web Service operations. The UML class diagram is shown in Figure 25.

RemoteDBWSWrapper
deleteResource(ResourceInfor[] Resources) getResources(Resourceinfor[] Resources) saveResources(GuiResource[] resources, boolean flag, boolean local, String host, String port)

Figure 25: RemoteDBWSWrapper.java

The Eclipse Web Tools Platform (WTP) was chosen as the development tool. The WTP extends the functionalities of the 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 to convert an existing Java class into a Web service. In addition, the latest Java JDK version 1.5 was used.

The UCLPv2 system uses the GT4 Java WS core service to host the UCLPv2 core Web Services. The downloading Web Service was implemented in this environment.

6.5. Class level design of resource advertising and discovery functionalities

Besides the downloading Web Service explained in Session 6.4, the following classes have been defined to provide functions for advertising a RDF-RL onto the public directory; discovering a RDF-RL; leasing and releasing a RL; cancelling the advertisement in a public Sesame directory. These functions utilize Sesame's Repository API to communicate with the public Sesame directory to perform their operations. The specification of the Sesame Repository API can be found in [28]. The relationship between these classes is shown in the UML class diagram in Figure 24.

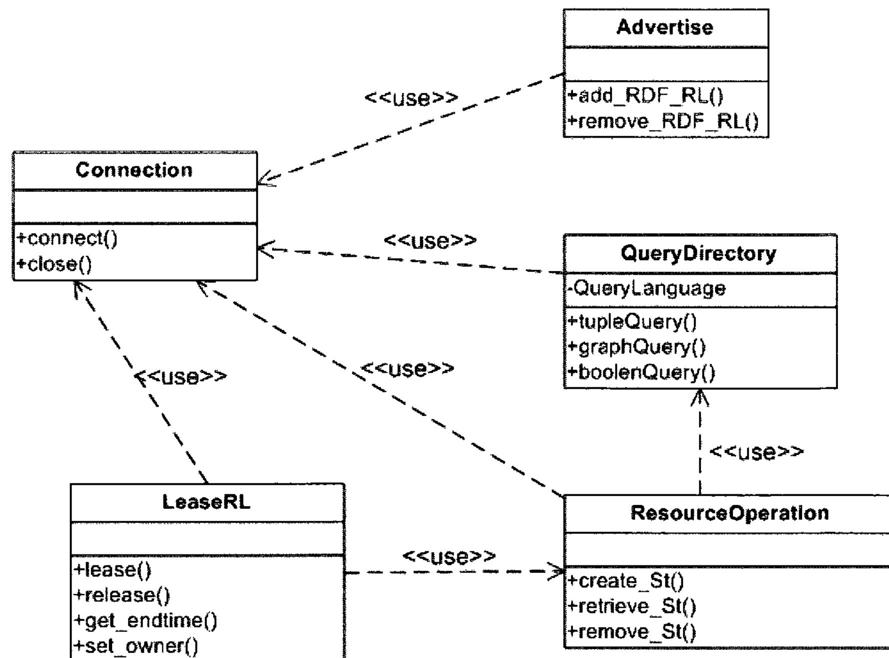

Figure 26: UML class diagram of Java classes for resource advertising and discovery classes

- Connection: This class uses Sesame's Repository API to define a connection method for connecting to a public Sesame directory. The connection method takes the directory's address and the ID of the repository as input to build the connection. An object of the class "Repository" from the package of `org.openrdf.repository` is return to represent the connection. This class also provides a close() method to disconnect from a public directory. Other classes use these connection or close methods.
- Advertise: This class defines a method to upload a RDF-RL to a public Sesame directory. A function to cancel an advertisement is defined in this class as well.
- QueryDirectory: This class provides a query method which takes a user query string as input and sends it to the public directory. It supports SPARQL, SeRQL and RQL query languages and three types of queries: the tuple query, the graph query and the boolean query. The result of a tuple query is a set of tuples (or variable bindings), where each tuple represents a solution of the query. This type of query is commonly used to get specific values (URIs, blank nodes, and literals) from the stored RDF data. The result of a graph query is an RDF graph (a set of RDF statements). The result of boolean queries is a simple boolean value, i.e. *true* or *false*. This type of query can be used to check if a repository contains specific information [28].

- ResourceOperation: this class manipulates individual RDF statements within RDF-RLs in a public Sesame directory. It provides methods to create, retrieve, and remove individual RDF statements. These methods are used to modify a RDF-RL's status such as change the lease values from false to true, or to indicate that the RDF-RL is leased by someone. If a public RDF-RL is being leased by someone, then the owner will be the organization who leased the RL. The client GUI uses this class to implement the resource discovery function.
- LeaseRL: this class will invoke the retrieve method of class ResourceOperation to get the end date of a given RDF-RL and will specify a lease end time that is no later than the end date of the RL in its lease method. Then it will change the owner of the RDF-RL to the leasing organization and the original owner to previous owner. The class also will invoke the RemoteDB_WS Web Service to download the original RL form the UCLPV2 server database. This class provides method to cancel the lease as well. The cancel operation involves changing the RDF-RL's current owner back to its original owner. Although the lease method has the ability of leasing individual resources from a RDF-RL, the UCLPv2 system only works at resource list level such as to create end-to-end connections. Therefore, the lease method currently only lets UCLPv2 users lease a RDF-RL.

6.6. Consideration of access right of the directory

The current UCLPv2 system assigns access rights to resources such as lightpaths, interfaces, and APNs via ownership. An organization is the owner of its resources and only the owner can perform operations on the resources. However, the ownership is changed when the resource list is advertised in the public directory. For example, a publicly advertised resource list has an owner called "public" which means that any organization other than the advertising organization can lease it and use it. Of cause, the owner of an advertised RL is changed to the importing organization when it is leased. We have assumed that only admin users of an UCLPv2 organization can advertise, discover and lease a RL in the public directory. Since the public directory may be open to users from other systems such as Chronos, DRAC or DRAGON, the public advertisement introduces a security risk of leasing resource lists to unintentional users, especially users that do not belong to UCLPv2 systems.

In order to ensure that only the right users can access the advertised resource lists, we propose a simple user name and password authentication mechanism: Each system such as UCLPv2, Chronos or DRAC needs to obtain a user account with a password that is created on the Sesame server. The Sesame server is responsible for verifying the user account and password. Each system needs to use its Sesame user account to access the Sesame directory.

6.7. Implementation and test

An UCLPv2 server has been installed on CAnet4 to let network users manage their lightpaths. The UCLPv2 server is running on a Linux machine with 2.4 Ghz Intel Core 2 CPU and 2 GigaByte memory. The RemoteDB-WS Web Service which provides the XML resource lists downloading function has been integrated and implemented on this UCLPv2 server. A Sesame server with version 2.2.4 has been installed on the same Linux machine as the UCLPv2 server. The running environment consists of Java 1.5.0_11, Apache Tomcat 5.5 and Fedora core 8. The UCLP RMC GUI is running on a client PC under Windows XP with Java 1.5.

6.7.1. Test of resource adverting, discovery and leasing

The use case described in Section 5.4.6 is used to demonstrate the success of the resource adverting, discovery and leasing. Three resource lists have been created by the organization CANARIE. The resource lists are "CANARIE-EC-001", "CANARIE-CRC-001" and "CANARIE-public-001". The resource list "CANARIE-EC-001" is exported to the organization called EC, the RL "CANARIE-CRC-001" is exported to the organization CRC and the last one is exported to public. After exporting these resource lists, the administrator of CANARIE launched the client GUI to advertise the three resources to the public Sesame directory. Figure 27 is the screen-shot of the GUI for adverting resource lists.

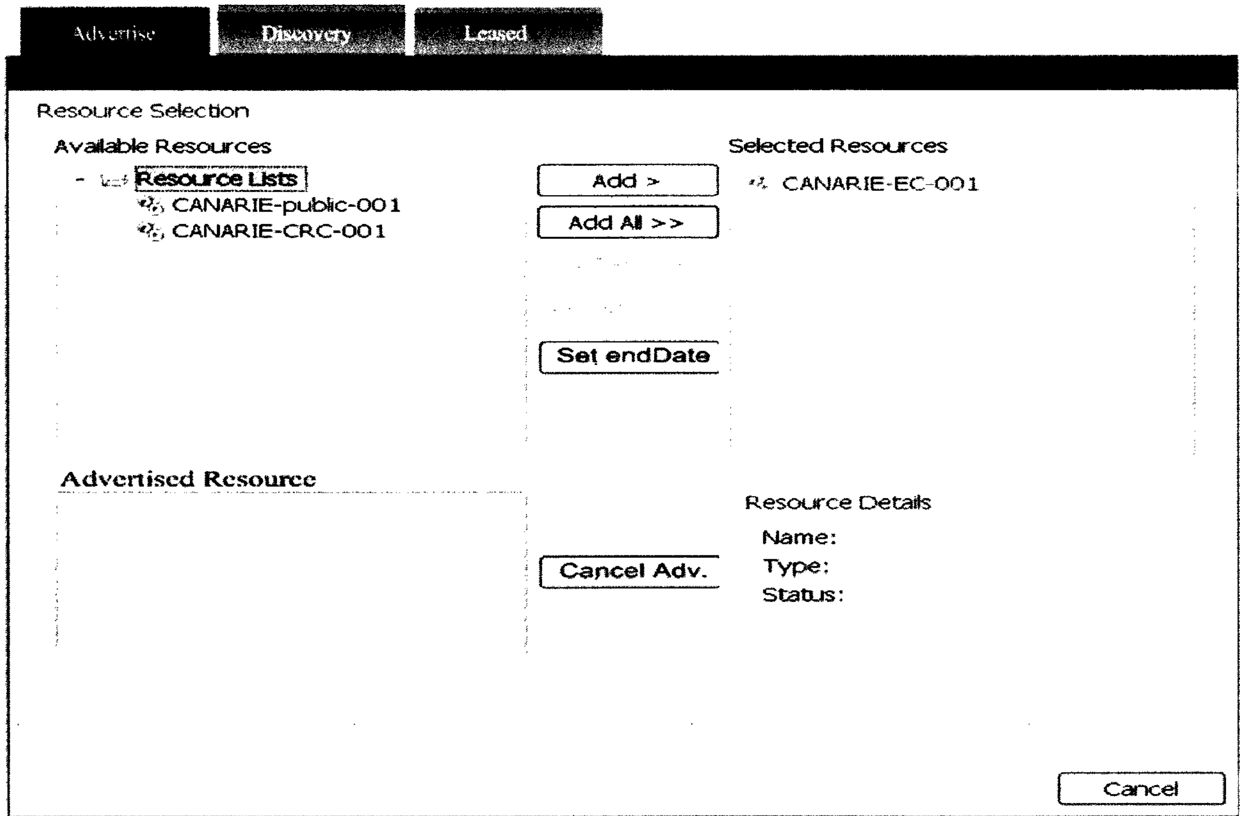

Figure 27: Resource lists adverting (screen-shot)

By using the console tool provided by the Sesame server, we can verify that the RDF-RLs were added to the public directory.

Next, the EC and CRC admin users launch their UCLPv2 RMC client GUI to discover the RDF-RLs in the public directory separately from different computers. Both EC and CRC admin users select the tag "Discover" and then the "Reserve" function to look for the resource list that was reserved for them. The EC admin user will see a RDF-RL with the name "CANAREI-EC-001" displayed in the GUI, whereas the CRC admin user will see that the "CANARIE-CRC-001" is reserved for CRC. Figure 28 shows the screen-shot of the GUI for discovering reserved resource lists for EC. Then the EC and CRC admin users decide to lease the reserved resource lists. First, the EC administrator selects the resource list. Next, by clicking the "Lease" button, the GUI will ask for a leasing end date. After the leasing end date is specified, the original XML resource list will be downloaded into the local workspace. The

value of the property "isLeased" is changed from "false" to "true" for both of the reserved RDF-RL in the directory.

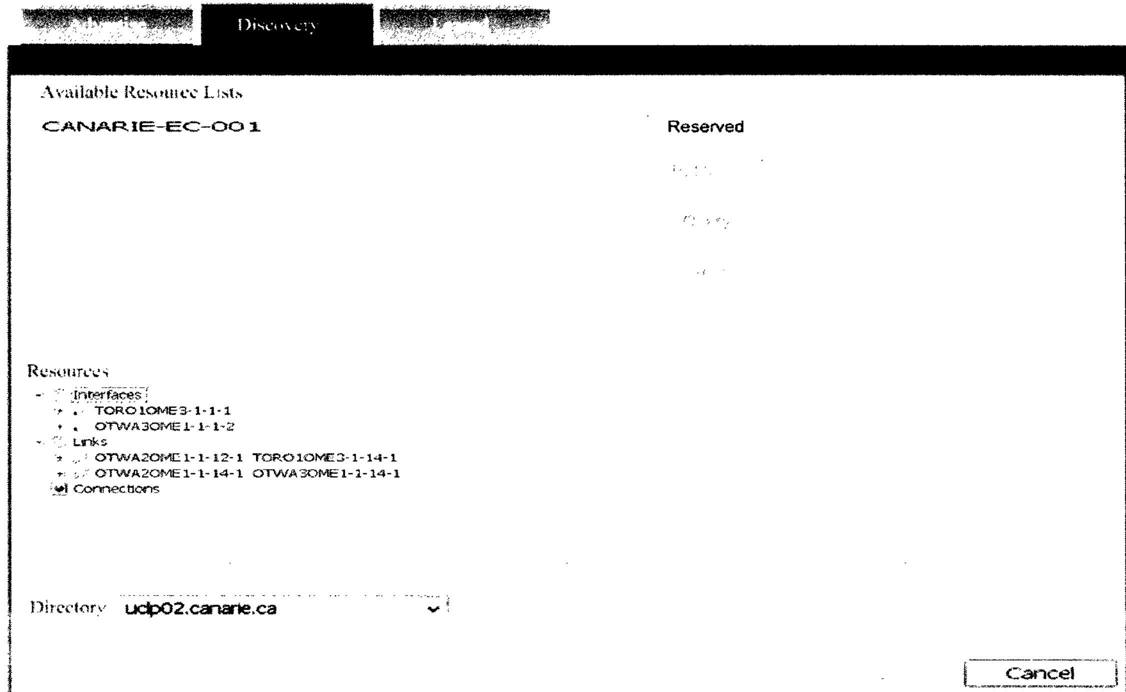

Figure 28: Resource discovery by reservation (screen-shot)

In order to look for the publicly advertised resource "CANARIE-public-001", either the admin user of EC and CRC can use the browse option to look into the public directory by clicking the "Public" button from the GUI. Both the admin users can see the "CANARIE-public-001" via the GUI. We assume that the EC admin user will lease the resource list. So the EC admin user selects the RDF-RL "CANARIE-public-001" and leases it.

By looking into the UCLPv2 RMC workspace of the EC admin user and the CRC admin user, we can verify that they have got the resource lists that were reserved for them and EC admin has got the publicly advertised resource list.

6.7.2. Performance test

In order to know the performance of the process of resource adverting and discovering in the CANet4 environment, we have defined two test cases. We assumed that for each lightpath provisioned, there would be a resource list being advertised and discovered in the Sesame directory. To reflect the process of adverting and discovering, we have tested the performance of the uploading and look-up processes of the Sesame directory in the two defined test cases in a test environment simulated the CANet4 networks, which typically has no more than 20 new lightpaths being provisioned each year and the number of lightpath remains within 120 for a year. We anticipated that the response time for the Sesame directory to answer a query would be within half a second. While for the uploading test, we would like to see the real result.

For the uploading test, we uploaded RDF-formatted resource lists to the Sesame directory and measured the time for the uploading process. The test dataset consists of 100 RDF-formatted resource lists which were evenly assigned to several organizations. A single RDF-formatted resource list contained 60 RDF statements with 5 devices, 5 locations, 12 interfaces and 4 lightpaths. The file size for a single resource list was around 5 Kbytes. The RDF-formatted resource list reflected a typical resource list in CANet4 network. The 100 RDF-formatted resource lists were uploaded to the Sesame directory one by one and the time for each round of uploading process was measured. The result is shown in Figure 29.

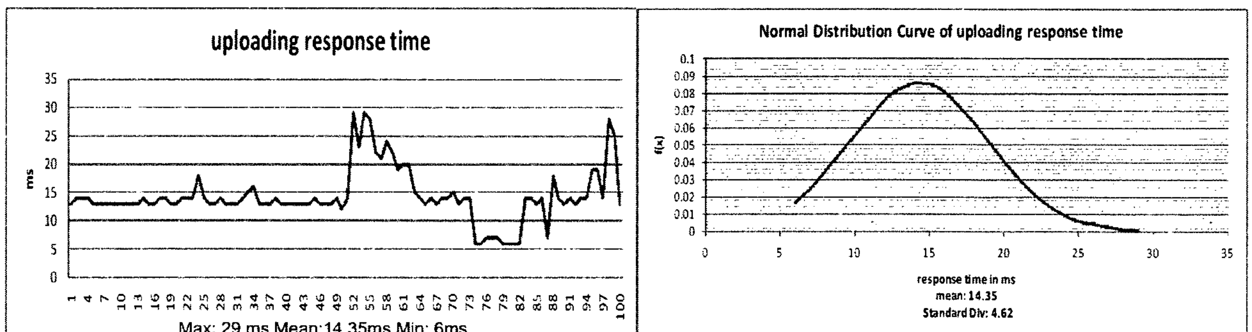

Figure 29: Uploading test result

As shown in Figure 29, the maximum, minimum and mean response time are 29ms, 6ms and 14.35ms respectively. We have derived the mean and standard deviation values from the data collected from the test result. By applying the normal distribution formulation, we could know the probability density of the distribution of the uploading time. From the normal distribution curve, applying the 3-σ rule (σ stands for standard deviation); we can say that in 99.7% of the cases the response time are smaller than 28.21ms. The result showed that the Sesame directory has a good performance for uploading.

For the look-up test, we timed the Sesame directory for its response to queries. In order to know how the look-up time depends on the number of resource lists in the Sesame directory, we performed three separate look-up tests with 100, 500 and 1000 resource lists in the Sesame directory. This is equivalent to having 3800, 19000 or 38000 RDF statements in the directory. We applied two types of queries for the look-up process. One type was to look for resource lists by ownership, while the other was looking for resource lists by range such as the device location, the interface and the lightpath bandwidth. We have sent these two queries to the Sesame directory alternatively for a total of 50 times for each test. The testing process was running on the same machine as the Sesame directory. We wanted to reduce the effect of the communication delay between the test process and the Sesame directory. The response time was measured at the client side. The test results are shown in Figure 30.

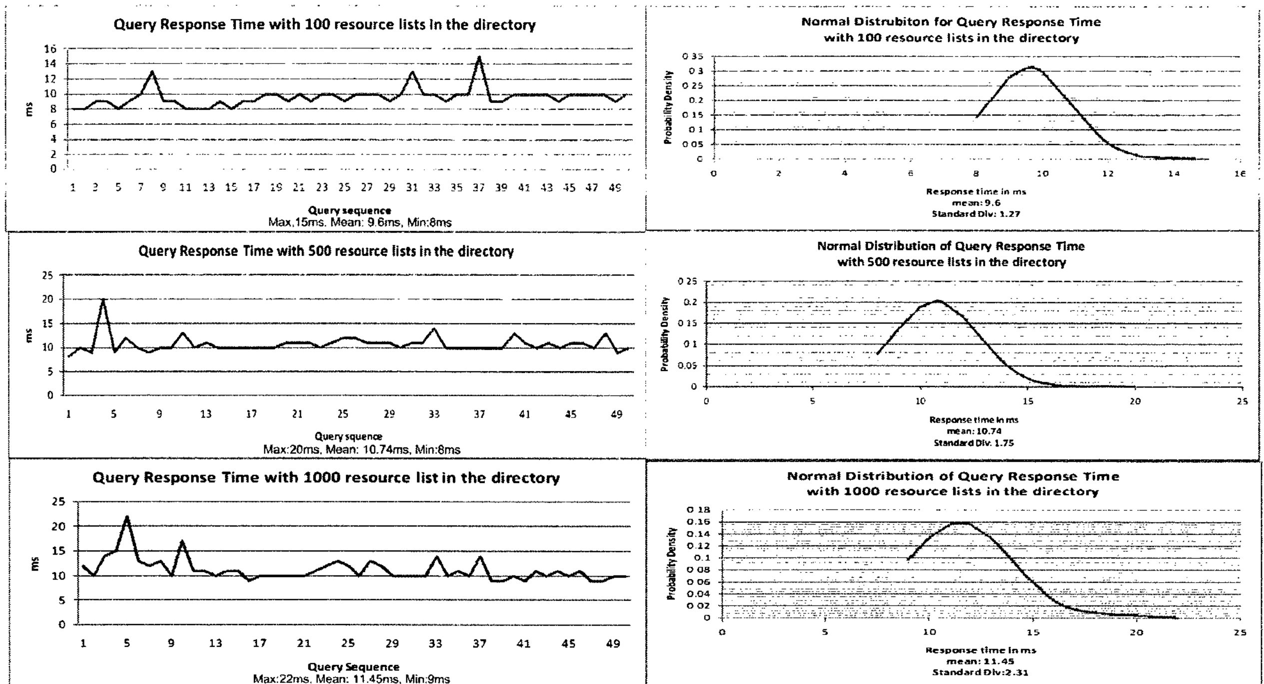

Figure 30 Test result for lookup response time

As shown in the figure, when the Sesame directory had 100 resource lists in the first look-up test, the maximum look-up time was 15ms, the minimum look-up time was 8ms and the average look-up time was 9.6ms. When the Sesame directory had 500 resource lists, the maximum and average look-up time was 20ms, 10.74ms, respectively, which were slightly higher than the corresponding look-up time in the first test. However, the minimum look-up time remained the same in both cases. When the Sesame directory had 1000 resource lists in the third test, the maximum, minimum and average look-up time were slightly higher than for the previous two tests. The maximum, minimum and average was 22ms, 9ms and 11.45ms, respectively. From the normal distribution curve derived from the three test result data, we could see that in 99.7% look-up time would be within 13.41ms for the first test, 15.99ms for the second test and 18.38ms for the third test.

In order to know how the knowledge-base system performs and to find supports to prove our testing result, we found a study [30] which tested four knowledge-base systems including a memory-based Sesame directory, with respect to use in large OWL applications. In this study, a total 1014 files were used which had a size ranging from 8MB to 583 MB for

the incrementally uploading test. And 15 queries had been designed and used for the query test. The total triples in the tested repository were around 6.8 millions. The result from this study showed that the memory-based Sesame directory was the best both for the uploading and query test when the data size was relatively small, e.g., 1 million triples. The result from the study supported our test result which proved that the memory-based Sesame directory is adequate and efficient for the use in CANet4 environment. In our test, the tested Sesame directory had occupied a size of 16MB of memory space with 60,000 triples. The size of the memory-based Sesame directory is only limited by the available memory space of the computer that hosts the Sesame directory.

Chapter 7 - Conclusion

The thesis describes the service-oriented architecture of the UCLPv2 system and the integration of a new functionality for resource advertising, discovery. The designed system uses a public Sesame directory and provides functionalities to allow UCLPv2 users to advertise and discover resources in the form of RDF-formatted resource lists. By using a Sesame directory as the public directory of the system and using RDF to describe the UCLP resource lists and resources, the designed system meets the first two system design requirements listed in section 5.2 which require the system not only to allow UCLP users to search resources based on resource properties and values, but also to allow UCLP users to perform range queries for finding all possible resources that may satisfy the user's requirements. The remaining system design requirements are met by the resource advertising and discovery functions designed and implemented as an extension of the UCLP system which allow UCLP users to advertise resources publicly or to reserve resources for specific users and to discover the advertised/reserved resources in the public Sesame directory. As an option, the designed resource adverting and discovery system provides a leasing function to allow UCLP users to lease the resources discovered in the public Sesame directory. The resource acquisition process is automated by using Web Service technology. Inter-working between UCLPv2 and other applications is achieved by adopting the NDL network topology schema in the description of resources and resource lists. The contributions of this thesis are the following:

- The design and implementation of the resource advertising and discovery mechanism for UCLPv2 systems using the Resource Description Framework (RDF) technology. The mechanism facilitates the UCLPv2 users to share resource and acquiring resources automatically. This is achieved by using a directory and implementing a new set of functions for resource advertising and discovery.
- As a network engineer for CANet4, I participated in the working group for inter-working among applications such as UCLPv2, DRAC and DRAGON. I explored the possibility to use the Resource Description Framework to describe UCLP resources

and resource lists and to utilize the advantages of ontology concepts for resource sharing among applications.

- I participated in the design and development of the Network Description Language (NDL) via the cooperation of GLIF organizations. I contributed to several RDF schema developments for NDL.
- I designed and implemented a Web Service that enables administrators of UCLPv2 organizations or other applications to download resource lists containing resources required by users. This automates the resource acquiring process. The Web Service is integrated into the UCLPv2 system.
- I designed and built a client GUI program which is integrated into the UCLPv2 client GUI which provides functions to allow administrators of UCLPv2 organizations to advertise and discover resource lists.

The resource advertising and discovery system presented in the thesis lays the foundation for future improvements, such as the following:

- One improvement would be to add more sophisticated security and access control mechanism for the resource advertising and discovery system, which should not be limited only to user account and password authentication, as used in the current system. This is more important for resource sharing among different applications.

- The UCLPv2 system does not work on individual resources. It requires a resource list as a unit of work. Consequently, the resources in an advertised resource list are only used by one organization. Users can not harvest individual resources. A future extension should allow UCLP users to harvest desired individual resources from one or several advertised resource lists and use the resources effectively.

- The inter-working between UCLPv2, DRAC, DRAGON and Chronos needs closer cooperation among the organizations that have developed and deployed these systems.

References

[1] "*Web Service Architecture*", D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, http://www.w3.org/TR/ws-arch/, February 2004. (Accessed in January, 2010)

[2] "*SOAP Version 1.2 Part 0: Primer (Second Edition)*", Nilo Mitra,Yves Lafon, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/ , April 2007. (Accessed in January, 2010)

[3] "*Web Services Description Language (WSDL) 1.1*", E.Christensen, F. Curbera, G. Meredith, S. Weerawarana, http://www.w3.org/TR/wsdl, March 2001. (Accessed in January, 2010)

[4] "*Web Services Resource Properties 1.2*", OASIS Standard, Steve Graham,Jem Treadwell, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf, April, 2006. (Accessed in January, 2010)

[5]"*UDDI Version 2.03 Data Structure Reference*", OASIS, Claus von Riegen, http://www.uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm, July 2002.(Accessed in January, 2010)

[6]"*UDDI Version 2.04 API Specification*", OASIS, Tom Bellwood, http://www.uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm, 19 July 2002. (Accessed in January, 2010)

[7] "*Jini Architecture specification*", www.jini.org, September, 2006. (Accessed in January 2010)

[8] "Jini Discovery and Join Specification", www.jini.org, September, 2006. (Accessed in January 2010)

[9] "*Jini Lookup Service Specification*", www.jini.org, September, 2006. (Accessed in January 2010)

[10] *Global Lambda Integrated Facilities*, http://www.glif.is, 2003. (Accessed in January, 2010)

[11] "*Harmony - Advanced reservations in Heterogeneous, Multi-domain environments*", A. Willner, C. Barz, S. Figuerola, J.A. García-Espín, J. Ferrer, and P. Martini, , in Proceedings of the IFIP-TC6 Networking '09, Springer Ed.

[12] "*Optical network design and planning*", Jane M. Simmons, New York:Springer, c2008 ISBN: 9780387764757 9780387764764 , 2008.

[13] *"Using the Network Description Language in Optical Networks"*, van der Ham, J. Grosso, P. van der Pol, R. Toonk, A. de Laat, C. IEEE, Integrated network management, May, 2007.

[14] *"Dynamic Resource Allocation Controller (DRAC)"*, Nortel networks, http://www.nortel.com/solutions/optical/collateral/nn110181.pdf, 2006. (Accessed in January, 2010).

[15] *"DRAGON: A framework for service provisioning in heterogeneous grid networks"*, T. Lehman, J. Sobieski, B. Jabbari, IEEE Communications Magazine 44 (3), March 2006.

[16] *"Importing the Sementic Web in UDDI"*, Massimo Paolucci , T. Kawamura , T. R. Payne , K. P. Sycara, Proc. of International Workshop on Web Services, E-Business, and the Semantic Web, p.225-236, ISBN:3-540-00198-0,May 2002.

[17] *"Evaluation of UDDI as a Provider of Resource Discovery Services for OGSA-based Grids"*, E. Benson, G. Wasson, M. Humphrey, IEEE, Parallel and Distributed Processing Symposium, 2006.

[18] *The Dublin Core Metadata Initiative*, http://dublincore.org/. (Accessed in December 2009)

[19] *"XML Schema Part 0: Primer"*, D. C. Fallside, P. Walmsley, http://www.w3.org/TR/xmlschema-0/, October 2004. (Accessed in January 2010)

[20] *"Introduce to XML schema"*, http://www.w3schools.com/Schema/schema_intro.asp, December 2009. (Accessed in January 2010)

[21]*"SPARQL Protocol WSDL 2.0 Description"*, Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana, http://www.w3.org/2007/SPARQL/protocol-query.wsdl, June 2007. (Accessed in January 2010)

[22] *"SPARQL Query Language for RDF"*, W3C recommandation, Eric Prud'hommeaux, Andy Seaborne, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/, January 2008. (Accessed in January 2010)

[23] *"User Guide for Sesame"*, Aduna B.V., http://www.openrdf.org/doc/sesame/users/userguide.html, 2006. (Accessed in January 2010)

[24] *Castor project*, http://www.castor.org/, November, 2009. (Accessed in January 2010)

[25] *"Customer-Managed End-to-End Lightpath Provisioning"*. J. Wu, M. Savoie, H. Zhang, S.Campbell, G. v. Bochmann and B. St. Arnaud . International Journal of Network Management.
Vol. 15, No. 5, Sept./Oct. 2005, pages: 349-362. Sell also *"What is UCLP"*, http://www.uclp.ca, January 2007. (Accessed in January 2010)

[26] *"UCLPv2 detailed design document"*, 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, http://uclp.ca/twiki/bin/viewauth/UCLP/DetailedDesignDoc , April 2006. (Accessed in January 2010)

[27]*"Development with JSP and XML-- Part II: JSP with XML in mind"*, Qusay H. Mahmoud, http://java.sun.com/developer/technicalArticles/xml/WebAppDev2/, 2001. (Accessed in January 2010)

[28] *"OpenRDF Sesame Core 2.3.0 API"*, http://www.openrdf.org/doc/sesame2/api/, 2006. (Accessed in January 2010)

[29]*"Web Services Addressing (WS-Addressing)"*, Don Box, Erik Christensen, Francisco Curbera, etc. http://www.w3.org/Submission/ws-addressing/. December 10, 2009. (Accessed in January 2010)

[30] *"An Evaluation of Knowledge Base Systems for Large OWL Datasets"*. Y. Guo, Z. Pan, and J. Heflin. Third International Semantic Web Conference, Hiroshima, Japan, LNCS 3298, Spinger (c), 2004, pp. 274-288

[31] *CANARIE Inc*, www.canarie.ca

[32] *"Advance and Immediate Reservations of Virtualized Network Resources"*, Laia Ferrao, Xavier Barrera, Eduard Grasa, Sergi Figuerola. TERENA Proceedings 2009, ISBN 978-90-77559-19-2

[33] *"Articulated Private Networks in UCLP"*, Eduard Grasa, Sergi Figuerola, Albert López, Gabriel Junyent, Michel Savoie, Bill St. Arnaud, Mathieu Lemay. Internet Research, Volume 17, Number 5, 2007, pages 535-545.

[34] *"Software Architectures for the Management and Control of User Owned and Sliceable Networks"*, Eduard Grasa Gras. Ph.D. thesis, PhD Program: Teoria del Senyal i Comunicacions Academic Year 2008-2009, UNIVERSITAT POLITÈCNICA DE CATALUNYA.

[35] *"Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema"*, Jeen Broekstra, Arjohn Kampman, Frank van Harmelen. The Semantic Web - ISWC 2002 In Proceedings of the first Int'l Semantic Web Conference (ISWC 2002), Vol. 2342 (29 May 2002), pp. 54-68.

Appendix A: RDF schema of UCLPv2 resource list

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:xsd="http://www.w3c.org/2001/XMLSchema#"
         xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:dcterms="http://purl.org/dc/terms/"
         xmlns:vs="http://www.w3.org/2003/06/sw-vocab-status/ns#"
         xmlns:ndl="http://www.science.uva.nl/research/sne/ndl#">

<rdf:Description rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#">
  <rdfs:isDefinedBy rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Ontology"/>
  <rdfs:label>resourcelist</rdfs:label>
  <dc:title xml:lang="en">UCLPv2 resource list module</dc:title>
  <dc:description xml:lang="en">
     A vocabulary for defining network resources and topologies on a single layer.
  </dc:description>
  <dc:publisher xml:lang="en">
     System and Network Engineering group, CANARIE Network
  </dc:publisher>
  <dcterms:issued>2009-08-26</dcterms:issued>
  <dcterms:modified>2009-10-21</dcterms:modified>
  <owl:versionInfo>1.1</owl:versionInfo>
  <vs:term_status>unstable</vs:term_status>
</rdf:Description>

<!-- Classes -->
<rdfs:Class rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#ResourceList">
    <rdfs:isDefinedBy
       rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
    <rdfs:label xml:lang="en">resource list</rdfs:label>
    <rdfs:comment xml:lang="en">
       A resource list contains network resources and some topology information.
    </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="ndl:Location">
     <rdfs:isDefinedBy
        rdf:resource="http://www.science.uva.nl/research/sne/schema/topology.rdf"/>
```

```
        <rdfs:label xml:lang="en">Location</rdfs:label>
        <rdfs:comment xml:lang="en">
          The location of a networdevice
        </rdfs:comment>
        <rdfs:subClassOf
          rdf:resource="http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing"/>
</rdfs:Class>

<rdfs:Class rdf:ID="ndl:Device">
      <rdfs:isDefinedBy
          rdf:resource="http://www.science.uva.nl/research/sne/schema/topology.rdf"/>
      <rdfs:label xml:lang="en">Device</rdfs:label>
      <rdfs:comment xml:lang="en">A (collection of) network element(s) that is grouped
          together representing a physical or abstracted network device.
       </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="ndl:Interface">
      <rdfs:isDefinedBy
          rdf:resource="http://www.science.uva.nl/research/sne/schema/topology.rdf"/>
      <rdfs:label xml:lang="en">Interface</rdfs:label>
      <rdfs:comment xml:lang="en">
          A network element(s) that can be represented as a connection point
           Typically a logical network interface.
       </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#Lightpath">
      <rdfs:isDefinedBy
          rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
      <rdfs:type rds:about="http://www.science.uva.nl/research/sne/ndl#Link"/>
      <rdfs:label xml:lang="en">Lightpath</rdfs:label>
      <rdfs:comment xml:lang="en">
          A concept of UCLPv2 lightpath
       </rdfs:comment>
</rdfs:Class>

<!-- Predicates -->
<rdf:Property rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#hasDevice">
      <rdfs:isDefinedBy
          rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
      <rdfs:label xml:lang="en">has device</rdfs:label>
      <rdfs:comment>
          The hasDevice property assigns devices to the resource list.
```

```
        </rdfs:comment>
        <rdfs:domain
            rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#ResourceList"/>
        <rdfs:range
            rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Device"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.science.uva.nl/research/sne/ndl#hasInterface">
        <rdfs:isDefinedBy
            rdf:resource="http://www.science.uva.nl/research/sne/schema/topology.rdf"/>
        <rdfs:label xml:lang="en">has interface</rdfs:label>
        <rdfs:comment>
            The hasInterface property assigns interfaces to the resource list.
        </rdfs:comment>
        <rdfs:domain
            rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#ResourceList"/>
        <rdfs:range
            rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Interface"/>
</rdf:Property>

<rdf:Property rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#hasLightpath">
        <rdfs:isDefinedBy
            rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
        <rdfs:label xml:lang="en">has lightpath</rdfs:label>
        <rdfs:comment>
            The hasDevice property assigns lightpath to the resource list
        </rdfs:comment>
        <rdfs:domain
            rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#ResourceList"/>
        <rdfs:range
            rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Lightpath"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.science.uva.nl/research/sne/ndl#connectedTo">
        <rdfs:isDefinedBy
            rdf:resource="http://www.science.uva.nl/research/sne/schema/topology.rdf"/>
        <rdfs:label xml:lang="en">connected to</rdfs:label>
        <rdfs:domain
            rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Lightpath"/>
        <rdfs:range
            rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Interface"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.science.uva.nl/research/sne/ndl#capacity">
```

```
    <rdfs:isDefinedBy
      rdf:resource="http://www.science.uva.nl/research/sne/schema/topology.rdf"/>
    <rdfs:label xml:lang="en">capacity</rdfs:label>
    <rdfs:comment>
      A float representing the data rate of an Interface or lightpath in Bytes/s
    </rdfs:comment>
    <rdfs:domain
      rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Lightpath"/>
    <rdfs:domain
      rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Interface"/>
    <rdfs:range
      rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.science.uva.nl/research/sne/ndl#locatedAt">
    <rdfs:isDefinedBy
      rdf:resource="http://www.science.uva.nl/research/sne/schema/topology.rdf"/>
    <rdfs:label xml:lang="en">located at</rdfs:label>
    <rdfs:comment>
      The relation between a Device and a Location
    </rdfs:comment>
    <rdfs:domain
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
    <rdfs:range
      rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Location"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.science.uva.nl/research/sne/ndl#name">
<!-- name is deprecated. You are recommended to use rdfs:label instead -->
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DeprecatedProperty"/>
    <owl:equivalentProperty rdf:resource="http://www.w3.org/2000/01/rdf-
      schema#label"/>
    <rdfs:label xml:lang="en">name</rdfs:label>
    <rdfs:comment>
      A short human-readable name for the subject
    </rdfs:comment>
    <rdfs:domain
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
     <rdfs:range
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
<rdf:Property rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#owner">
    <rdfs:isDefinedBy
rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
```

```
      <rdfs:label xml:lang="en">owner</rdfs:label>
      <rdfs:comment>current owner of the resource list.</rdfs:comment>
      <rdfs:domain
          rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#ResourceList"/>
       <rdfs:range
         rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#pre-owner">
      <rdfs:isDefinedBy
rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
      <rdfs:label xml:lang="en">pre-owner</rdfs:label>
      <rdfs:comment>previous owner of the resource list.</rdfs:comment>
      <rdfs:domain
         rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#ResourceList"/>
      <rdfs:range
         rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://www.science.uva.nl/research/sne/ndl#description">
      <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DeprecatedProperty"/>
      <owl:equivalentProperty rdf:resource="http://purl.org/dc/elements/1.1/description"/>
      <rdfs:label xml:lang="en">description</rdfs:label>
      <rdfs:comment>A longer human-readable description for the subject.</rdfs:comment>
      <rdfs:domain
         rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
      <rdfs:range
         rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#endDate">
      <rdf:type rdf:resource="http://www.w3c.org/2001/XMLSchema#dateTime"/>
      <rdfs:label xml:lang="en">endDate</rdfs:label>
      <rdfs:comment>advertise end date and time</rdfs:comment>
      <rdfs:domain
         rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#ResourceList"/>
      <rdfs:range
         rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#type">
      <rdfs:isDefinedBy
rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
      <rdfs:label xml:lang="en">type</rdfs:label>
```

```
    <rdfs:comment>indicate the interface technology</rdfs:comment>
    <rdfs:domain
      rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Interface"/>
    <rdfs:range
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#isLeased">
    <rdfs:isDefinedBy
rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
    <rdfs:label xml:lang="en">isLeased</rdfs:label>
    <rdfs:comment>current status of the resource list.</rdfs:comment>
    <rdfs:domain
      rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#ResourceList"/>
    <rdfs:range
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#direction">
    <rdfs:isDefinedBy
      rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
    <rdfs:label xml:lang="en">direction</rdfs:label>
    <rdfs:comment>indicate lightpath type. 2way or 1way</rdfs:comment>
    <rdfs:domain
      rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#Lightpath"/>
    <rdfs:range
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:about="http://uclp02.canarie.ca/schema/rdf/resourcelist#wsAccessURL">
    <rdfs:isDefinedBy
rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist.rdf"/>
    <rdfs:label xml:lang="en">wsAccessURL</rdfs:label>
    <rdfs:comment>resource list downloading Web Service address</rdfs:comment>
    <rdfs:domain
      rdf:resource="http://uclp02.canarie.ca/schema/rdf/resourcelist#ResourceList"/>
    <rdfs:range
      rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

</rdf:RDF>
```

Appendix B: An example of a simple network described by NDL

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ndl="http://www.science.uva.nl/research/air/ndl#">

<ndl:Location rdf:about="#Lighthouse">
        <ndl:name>Lighthouse</ndl:name>
</ndl:Location>

<ndl:Device rdf:about="#Rembrandt3">
        <ndl:name>Rembrandt3</ndl:name>
        <ndl:locatedAt rdf:resource="#Lighthouse"/>
        <ndl:hasInterface rdf:resource="#Rembrandt3:eth0"/>
        <ndl:hasInterface rdf:resource="#Rembrandt3:eth1"/>
</ndl:Device>

<ndl:Interface rdf:about="#Rembrandt3:eth0">
        <ndl:name>Rembrandt3:eth0</ndl:name>
        <ndl:capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
        1.2E+9
        </ndl:capacity>
        <ndl:connectedTo rdf:resource="#Speculaas:port3"/>
        </ndl:Interface>

<ndl:Interface rdf:about="#Rembrandt3:eth1">
        <ndl:name>Rembrandt3:eth1</ndl:name>
        <ndl:capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
        1.2E+9
        </ndl:capacity>
        <ndl:connectedTo rdf:resource="#lp-specuRemb"/>
</ndl:Interface>

<ndl:Device rdf:about="#Speculaas">
        <ndl:name>Speculaas</ndl:name>
        <ndl:locatedAt rdf:resource="#Lighthouse"/>
        <ndl:hasInterface rdf:resource="#Speculaas:port3"/>
        <ndl:hasInterface rdf:resource="#Speculaas:port4"/>
</ndl:Device>

<ndl:Interface rdf:about="#Speculaas:port3">
        <ndl:name>Speculaas:port3</ndl:name>
        <ndl:capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
        1.2E+9
```

```
        </ndl:capacity>
        <ndl:connectedTo rdf:resource="#Rembrandt3:eth0"/>
 </ndl:Interface>

<ndl:Interface rdf:about="#Speculaas:port4">
        <ndl:name>Speculaas:port4</ndl:name>
        <ndl:capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
        1.2E+9
        </ndl:capacity>
        <ndl:connectedTo rdf:resource="#lp-specuRemb"/>
 </ndl:Interface>

<ndl:link rdf:about="#lp-specuRemb">
        <ndl:name>lp-specuRemb</ndl:name>
        <ndl:capacity rdf:datatype="http://www.w3.org/2001/XMLSchema#float">
        1.2E+9
        </ndl:capacity>
        <ndl:connectedTo rdf:resource="#Rembrandt3:eth1"/>
        <ndl:connectedTo rdf:resource="#Speculaas:eth1"/>
</ndl:Link>

</rdf:RDF>
```

Appendix C: WSDL for UCLPv2 RemoteDB-WS

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://core.inocybe.ca/2007/08/GuiRemoteDB-WS"
    targetNamespace="http://core.inocybe.ca/2007/08/GuiRemoteDB-WS"
    name="GuiRemoteDB-WS">
    <wsdl:types>
        <xsd:schema>
            <xsd:import namespace="http://core.inocybe.ca/2007/08/GuiRemoteDB-WS"
                schemaLocation="GuiRemoteDB-Types.xsd">
            </xsd:import>
        </xsd:schema>
        <xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
             targetNamespace="http://core.inocybe.ca/2007/08/GuiRemoteDB-WS">
             <xsd:element name="removeLock_fault" type="xsd:string">
             </xsd:element>
       </xsd:schema>
    </wsdl:types>
    <wsdl:message name="SaveResourcesResponse">
        <wsdl:part element="tns:SaveResourcesResponse"
            name="SaveResourcesResponse" />
    </wsdl:message>
    <wsdl:message name="SaveResourcesRequest">
        <wsdl:part name="SaveResourcesRequest"
            element="tns:SaveResourcesRequest" />
    </wsdl:message>
    <wsdl:message name="SaveResourcesFault">
        <wsdl:part name="SaveResourceFault"
            element="tns:SaveResourcesException">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="getFileHistResponse">
        <wsdl:part element="tns:getFileHistResponse"
            name="getFileHistResponse" />
    </wsdl:message>
    <wsdl:message name="getFileHistRequest">
        <wsdl:part name="getFileHistRequest"
            element="tns:getFileHistRequest" />
    </wsdl:message>
    <wsdl:message name="getFileHistFault">
```

```
        <wsdl:part name="getFileHistFault"
            element="tns:getFileHistException">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="QueryAvailableResourcesRequest">
        <wsdl:part name="QueryAvailableResourcesRequest"
            element="tns:QueryAvailableResourcesRequest">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="QueryAvailableResourcesResponse">
        <wsdl:part name="QueryAvailableResourcesResponse"
            element="tns:QueryAvailableResourcesResponse">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="QueryAvailableResourcesFault">
        <wsdl:part name="QueryAvailableResourcesFault"
            element="tns:QueryAvailableResourcesException">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="GetResourcesResponse">
        <wsdl:part name="GetResourcesResponse"
            element="tns:GetResourcesResponse">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="GetResourcesRequest">
        <wsdl:part name="GetResourcesRequest"
            element="tns:GetResourcesRequest">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="GetResourcesFault">
        <wsdl:part name="GetResourcesFault"
            element="tns:GetResourcesException">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="DeleteResourcesRequest">
        <wsdl:part name="DeleteResourcesRequest"
            element="tns:DeleteResourcesRequest">
        </wsdl:part>
    </wsdl:message>
    <wsdl:message name="DeleteResourcesResponse">
        <wsdl:part name="DeleteResourcesResponse"
            element="tns:DeleteResourcesResponse">
        </wsdl:part>
    </wsdl:message>
```

```
<wsdl:message name="DeleteResourcesFault">
     <wsdl:part name="DeleteResourcesFault"
          element="tns:DeleteResourcesException">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="ValidateNewResourceRequest">
     <wsdl:part name="ValidateNewResourceRequest"
          element="tns:ValidateNewResourceRequest">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="ValidateNewResourceResponse">
     <wsdl:part name="ValidateNewResourceResponse"
          element="tns:ValidateNewResourceResponse">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="ValidateNewResourceFault">
     <wsdl:part name="ValidateNewResourceFault"
          element="tns:ValidateNewResourceException">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="GetOrganizationsRequest">
     <wsdl:part name="GetOrganizationsRequest"
          element="tns:GetOrganizationsRequest">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="GetOrganizationsResponse">
     <wsdl:part name="GetOrganizationsResponse"
          element="tns:GetOrganizationsResponse">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="GetOrganizationsFault">
     <wsdl:part name="GetOrganizationsFault"
          element="tns:GetOrganizationsException">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="removeLockRequest">
     <wsdl:part name="removeLockRequest" element="tns:RemoveLockRequest">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="removeLockResponse">
     <wsdl:part name="removeLockResponse" element="tns:RemoveLockResponse">
     </wsdl:part>
</wsdl:message>
<wsdl:message name="removeLock_faultMsg">
```

```
        <wsdl:part name="fault" element="tns:RemoveLockFault">
        </wsdl:part>
    </wsdl:message>
    <wsdl:portType name="GuiRemoteDB-WSPortType">
        <wsdl:operation name="saveResources">
            <wsdl:input message="tns:SaveResourcesRequest" />
            <wsdl:output message="tns:SaveResourcesResponse" />
            <wsdl:fault name="fault" message="tns:SaveResourcesFault">
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getFileHistory">
            <wsdl:input message="tns:getFileHistRequest" />
            <wsdl:output message="tns:getFileHistResponse" />
            <wsdl:fault name="fault" message="tns:getFileHistFault">
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="queryAvailableResources">
            <wsdl:input message="tns:QueryAvailableResourcesRequest"></wsdl:input>
            <wsdl:output
                message="tns:QueryAvailableResourcesResponse">
            </wsdl:output>
            <wsdl:fault name="fault"
                message="tns:QueryAvailableResourcesFault">
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getResources">
            <wsdl:input message="tns:GetResourcesRequest"></wsdl:input>
            <wsdl:output message="tns:GetResourcesResponse"></wsdl:output>
            <wsdl:fault name="fault" message="tns:GetResourcesFault"></wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="deleteResources">
            <wsdl:input message="tns:DeleteResourcesRequest"></wsdl:input>
            <wsdl:output message="tns:DeleteResourcesResponse"></wsdl:output>
            <wsdl:fault name="fault"
                message="tns:DeleteResourcesFault">
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="validateNewResource">
            <wsdl:input message="tns:ValidateNewResourceRequest"></wsdl:input>
            <wsdl:output message="tns:ValidateNewResourceResponse"></wsdl:output>
            <wsdl:fault name="fault"
                message="tns:ValidateNewResourceFault">
            </wsdl:fault>
        </wsdl:operation>
```

```
        <wsdl:operation name="getOrganizations">
            <wsdl:input message="tns:GetOrganizationsRequest"></wsdl:input>
            <wsdl:output message="tns:GetOrganizationsResponse"></wsdl:output>
            <wsdl:fault name="fault"
              message="tns:GetOrganizationsFault">
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="removeLock">
            <wsdl:input message="tns:removeLockRequest">
            </wsdl:input>
            <wsdl:output message="tns:removeLockResponse">
            </wsdl:output>
            <wsdl:fault name="fault" message="tns:removeLock_faultMsg">
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:portType>
</wsdl:definitions>
```