A Service Oriented Architecture for Deploying and Managing Network Services

Victor A.S.M. de Souza and Eleri Cardozo*

Department of Computer Engineering and Industrial Automation,
School of Electrical and Computer Engineering,
State University of Campinas,
13083-970, Campinas, So Paulo, Brazil
{vsouza, eleri}@dca.fee.unicamp.br

Abstract. New generation network services must be deployed and managed according to the customers' specific requirements. In this context, service providers must devise a way to design network services with near zero development time and high degrees of customization and evolution. Customization is necessary to fit the service according to the customers' requirements, while evolution is necessary to adapt the service as soon as these requirements change. In addition, customers are demanding the ability to manage the service in order to keep the usage, configuration, and evolution under their control. This paper presents an approach based on service oriented architecture (SOA) for developing network services able to fulfill the requirements of rapid deployment, customization, and customer-side manageability. The approach considers the network service as a set of interacting elements implemented as Web Services. The service logic is expressed in terms of Web Services orchestration. Two services for the management of connections in optical networks are presented as a case study.

1 Introduction

New network services differ from the present ones in many aspects. Firstly, the development time of the new services must be kept close to zero. In other words, the time between the service design and its effective use must be very short (ideally zero, no more than few hours in special cases). Secondly, the service must take into account the exact customers' requirements (and expectations) such as aspects related to configuration, pricing, and quality. Finally, customers are demanding the ability to manage the main aspects of the service in order to take advantage of their business peculiarities, e.g., traffic patterns and end-user profiles. Of course, price is always an important variable that can be reduced as long as the complexities of service creation, deployment and management decrease.

In this scenario, network providers must devise new ways of designing, deploying and managing network services. We strongly believe that service composition is the key toward this objective. A network service can be created by composing a set of primitive services. This recurrent definition is important in the sense that

* The authors would like to thank Ericsson Brazil for its support.

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 465–477, 2005.

it allows a more complex service be built above a set of already existing services such as back-end, resource management, and network management services. For instance, a Virtual Private Network (VPN) service can be built by composing a connection service, an authentication service, a fault management service, and a resource management service. These composed services are distributed throughout the service provider's enterprise, some of them running on the central office and others running close to the transport network.

Actually, there is a gap between the software entities at the business level and at the network level. Entities at the network level are based on low level signaling protocols such as RSVP-TE (Resource Reservation Protocol - Traffic Engineering) and OIF's UNI (Optical Internetworking Forum's User-to-Network Interface). The access to these protocols are performed via operator's interface, network management protocols (e.g., SNMP - Simple Network Management Protocol), or proprietary application programming interfaces (APIs). The entities at the business level, on the other hand, rely on high level software artifacts such as enterprise components (e.g., Enterprise Java Beans, COM+) and web components (e.g., Java Server Pages, Active Server Pages). Clearly, the gap between the control and management entities at network level and the entities at the business level is a complicating factor when a network service with high level of automation and integration must be designed and deployed in a short period of time.

Service oriented computing (SOC) is an attractive solution to narrow this gap in the sense that it offers a good level of automation, integration, customization, and flexibility in service creation, deployment, and management. This paper proposes a service oriented architecture (SOA) for management and deployment of services in a connection oriented network. The architecture assumes that all the composed services are Web Services and composition is governed by Web Services orchestration and choreography techniques. In this architecture, service creation consists in the edition of an orchestration script while the service deployment consists in the installation of this script in an appropriated software engine. Service management is described via Web Services choreography, an interaction agreement between different organizations or independent processes, in our case service user and service provider. The edition of the orchestration script can be assisted by specialized service creation tools, general purpose text editors or even by software engineering modeling tools.

This paper is organized as follows. Section 2 presents a brief introduction to SOC. Section 3 presents the proposed SOA-based architecture for service creation, deployment, and management focusing on connection oriented networks. Section 4 presents the services developed for assess the proposed architecture. Section 5 discusses some related and recently published works. Finally, Sect. 6 presents some closing remarks.

2 Service Oriented Computing

Service oriented computing is considered by many a step forward in the distributed computing field. Distributed computing, mainly distributed objects and,

most recently, components, provide high cohesion, lower coupling, and modularity to the applications. As a consequence, software reuse and evolution are favored, with reduction of development and maintenance costs. Several open standards for distributed computing such as CORBA (Common Object Request Broker Architecture) and CCM (CORBA Component Model) from OMG (Object Management Group), RMI (Remote Method Invocation) and EJB (Enterprise Java Beans) from the Java Community are mature and well accepted. Software platforms supporting these standards are available, both commercially and as open source software.

Unfortunately, these standards and platforms have not been used across enterprises. Security and interoperability issues are the most relevant reasons. Protocols such as IIOP (Internet Inter-ORB Protocol) employed in CORBA and RMI are not "firewall friendly", while the interoperability between different standards has never been completely addressed. Moreover, loose-ends on the standards may result in interoperability problems between platforms implementing the same standard.

It was in this context that SOC emerged, providing a way to interoperate large software entities, independently of software platforms and systems employed by each enterprise. Service oriented computing is defined as "the computing paradigm that utilizes services as fundamental elements for developing applications" [1]. Using orchestration mechanisms we can build a more comprehensive service that can itself be part of a higher level composition [2]. The service workflow is defined in orchestration scripts that is processed by an orchestration engine.

Some relevant characteristics of SOC reported in the literature are summarized below:

- Interoperability - achieved through the use of an independent transport protocol.
- Composability - services can be composed to form another service, providing a flexible, rapid and low cost way of creating new services.
- Reusability - being a modular unit of software, services generally can be reused, reducing the time and effort to build new applications.
- Ubiquity - services can be accessed from anywhere at any time.
- Granularity - services presents higher granularity when compared with objects and components, facilitating the development of complex systems.
- Coupling - loose coupling is achieved in different levels:
  - Just real dependencies among software elements are implemented as long as each service has its own defined interface;
  - Using registering and discovering mechanisms no coupling is made to the service location;
  - Platform and language coupling can be avoided using a platform independent transport protocol;
  - Synchronism due to the request-response invocation style can be attenuated using asynchronous message exchange.

As mentioned in Sect. 1, our objective is to enable a quick and flexible development of new network services in connection oriented networks by addressing interoperability and high coupling issues both inside and outside an administrative network domain. The mentioned SOC characteristics fullfill these requirements. The most relevant consequence of employing SOC in this field is a complete integration of the network and business layers. This integration was proposed more than a decade ago in the scope of TMN (Telecommunication Management Network). However, TMN never achieved such an integration due to the need of gateways (mediation and adaptation functions in TMN) to connect software entities at different layers.

3 The Service Oriented Control and Management Architecture

3.1 Architecture Overview

In SOA every logical entity is seen as a service. Services are built from scratch or from legacy software by adding appropriated wrappers. These existing services can be classified into two levels, the business and the network levels. Services at the business level relate to the enterprise itself (e.g., subscription), while services at the network level relate to the transport network (e.g., routing). As stated before, there is a gap between services located into these two levels in terms of software entities the services are based on. A common approach is to employ a gateway software that converts the high level business decisions to the low level network signaling. This is not a good solution as gateways are always bottlenecks for interoperability, reliability and performance. Moreover, this approach ties too hard the business and network layers, compromising service automatization, customization and evolution.

In order to avoid gateways, we propose to enhance the network elements such as routers and switches with a service interface. In our architecture, every network element offers control and management functions through Web Services. Since the network element already hosts a web server, extending this server toward a Web Service engine is a minor step without significant impact to the equipment final cost. In addition, to allow a smooth integration between the business and network layers, the service interface of the network equipments may eliminate the need of complex network protocols. In our case, the connection establishment protocol (usually RSVP-TE) is replaced by a much simpler orchestration script. In this way we build a single service-based bus of communication, where one central orchestrated service is responsible for receiving client requests and coordinate all calls to the composed services. This architecture is presented in Fig. 1. The orchestrated service exposes a Web Service interface to the service client.

3.2 Service Management

Once installed, the network service provides the client the ability to manage the service, for instance, to alter the topology of a Virtual Private Network.

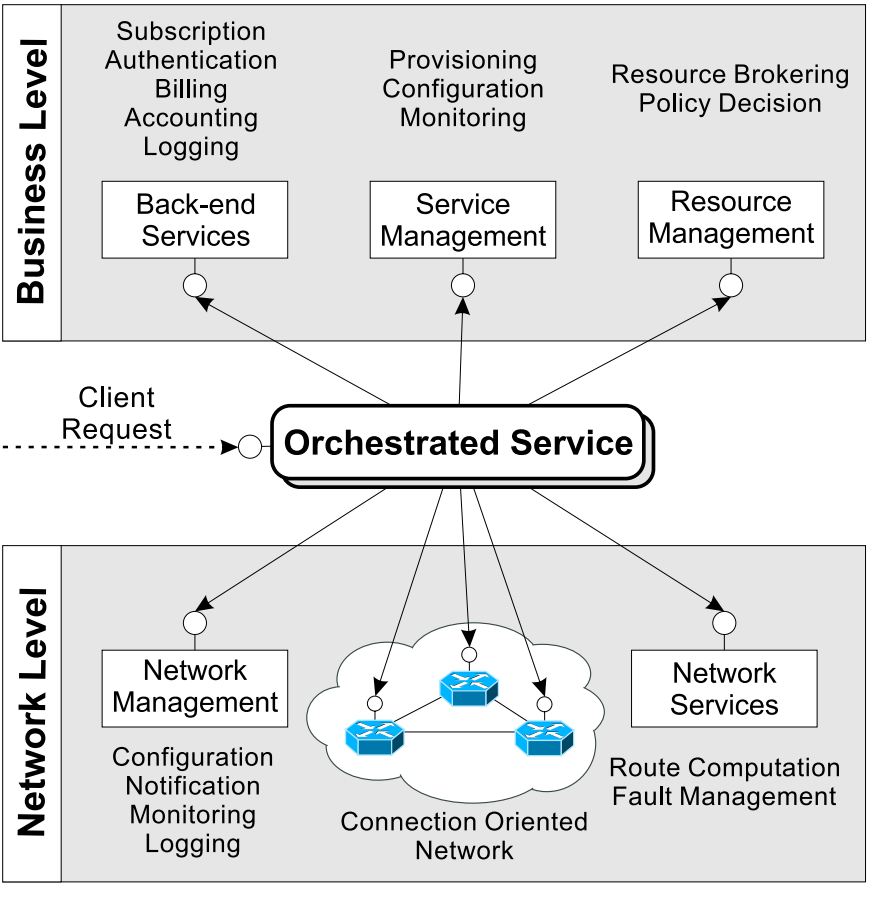

Fig. 1. Proposed architecture

The network service can be accessed via user interfaces (e.g., web interfaces) or via programatic interfaces where the service is accessed by other applications. This allows the composition of end-to-end service over multiple domains, e.g., VPN services, Voice-over-IP (VoIP) services or Virtual LAN (VLAN) services. In this context, it is necessary to precisely and unambiguously describe the collaborations between the client and the service provider. This is exactly what the choreography mechanism proposes [3]. As such, the service interface provided to the client will have their interaction contract described by a choreography script as shown in Fig. 2.

3.3 Service Creation

Service creation is a matter of composing the services with specific functions available at the business and the network levels. The hardest way to create an orchestrated network service is by manually editing an orchestration script.

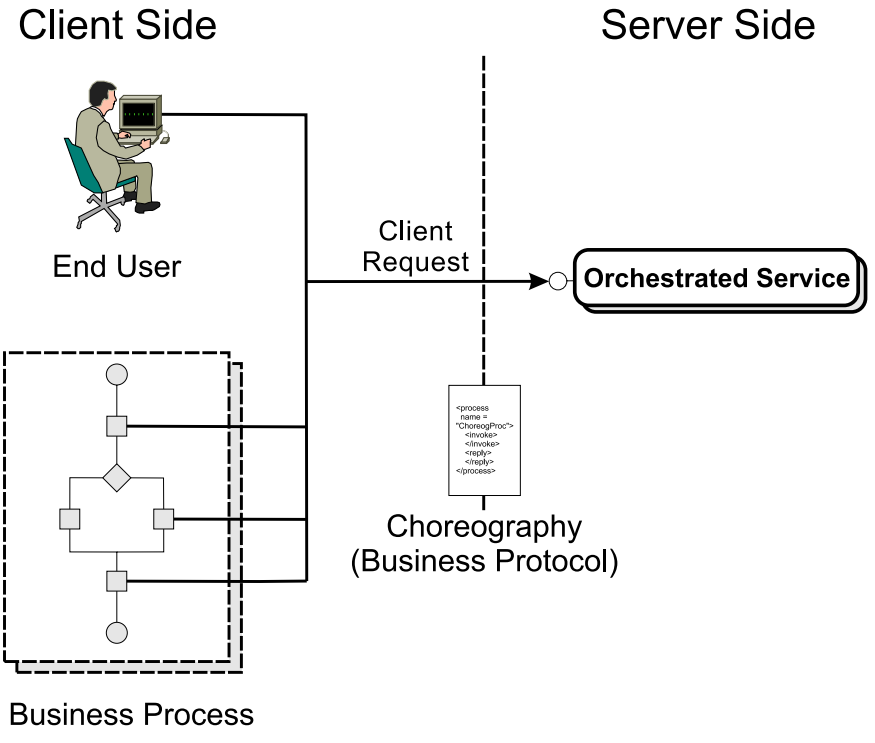

Fig. 2. Proposed client architecture

Another possibility is to employ a general modeling language such as UML (Unified Modeling Language) to specify the interactions among the composed service, using, for instance, an UML activity diagrams [4]. For this, it is necessary to specify a new UML profile in order to represent the particular domain of interest. It is also necessary to use model transformations to translate the UML model to the target orchestration language.

The most convenient way to create services is to use a Service Creation Environment (SCE), a software designed specially to this task [5, 6]. SCEs employ dedicated graphical interfaces with terms, icons, and diagrams known to the network engineer. The output of the SCE is exactly the orchestration script ready to execute in an orchestration engine. In any of the presented possibilities, service composition leads to a reduction in the service development time, a major issue in today's dynamic business environments.

3.4 Service Deployment

Using orchestration, the service deployment consists simply in the installation of the script in the orchestration engine. This can be performed via programatic interfaces (e.g., invoked by the SCE) or manually, via a graphical user interface provided by the orchestration engine.

4 Implementation Description

The architecture proposed in the Sect. 3 is general enough to deploy network services of arbitrary complexity. As a case study, we have developed two optical network services based on the proposed architecture: an Optical Connection Service (OCS) and a Fault Management Service (FMS).

As in any connection oriented network, in optical networks it is necessary to establish a connection before the traffic can be sent over the network. In optical networks, connections are lightpaths that may transverse a number of optical switches. The objective of the Optical Connection Service is to enable network clients to create, manage and destroy lightpaths. The Fault Management Service is aimed at restoring failed connections by setting up protection lightpaths.

Each optical switch in the domain will expose a cross connection service, enabling the composition engine to set up the connections across the domain. As the time required to setup a cross connection inside an optical switch is high (order of *ms*) it is expected that the Web Service does not introduce a considerable overhead in the connection establishment time.

All other necessary services to provide the connection service could be installed anywhere in the network, even on a third party domain (e.g., the billing service could be provided by a credit card operator).

4.1 Composed Services

We identified the following composed services necessary to implement the Optical Connection Service and the Fault Management Service:

- Optical Switch Service (OSS): a service exposed by the optical switch used to cross connect ports (fibers), wavelengths or wavebands.
- Resource Management Service (RMS): a service responsible for storing data about installed lightpaths, available resources, and Service Level Agreements (SLAs). It also stores protection information associated to a lightpath.
- Routing Service (RS): a service responsible for computing a route inside the optical network, given an ingress node, an egress node, and the available resources on the network. It is also responsible for finding a disjoint route for protection paths. The service must run a Routing and Wavelength Assignment (RWA) algorithm over a topology discovered with the aid of a routing protocol such as Open Shortest Path First - Traffic Engineering (OSPF-TE).
- Authentication Service (AS): a service responsible for authenticating the network users.
- Accounting Service (AcS): a service responsible for keeping track of the resources effectively used by a client for accounting and billing purposes.
- Logging Service (LS): a service responsible for logging errors and any other relevant information.

4.2 Implementation Details

In our system, the OSS, RMS, RS, AS, and LS composed services as previously described were implemented from scratch. In a service provider environment,

most of the composed services already exist, although not as Web Services. To expose an existing service as a Web Service, the developer can take advantage of many existing software tools. For instance, some EJB platforms generate Web Service interface for existing enterprise beans (components).

We chose the Business Process Execution Language for Web Services (BPEL) [7], as it is the most mature standard language for orchestration and choreography (through the use of BPEL's *Executable Processes* and *Business Protocols*, respectively). As orchestration engine we chose ActiveBPEL [8] because it is open source and implements all the BPEL 1.1 specification including the full complement of BPEL activities.

Using the compensation mechanism provided by the BPEL language we are able to support rollback in connection establishment. This mechanism defines how individual or composite activities within a process are to be compensated in cases where exceptions occur during service invocations. When an optical switch in the connection path, for any reason, cannot install the cross connection as required, the previous switches that have already set their internal cross connections need to undo the action previously taken. The orchestration engine is in charge of coordinating the compensation actions that must be taken (connection release in this case).

All the topology information about the optical network is stored in a database. We have implemented a web interface where the network administrator can set the physical network topology, including nodes, fibers, wavelengths per fiber, cost of fiber, switching capacity, among other informations. This web interface is shown in Fig. 3. In a real optical network this information could be discovered using a routing algorithm such as OSPF-TE, but this is not relevant for the evaluation of the connection and fault management services.

The RMS is in charge of keeping this database up-to-date. Before any resources can be effectively used the RMS must be notified. This is necessary in order to keep the network state up-to-date. The RS is capable of finding a route inside the network, running a Shortest Path First (SFP) algorithm. It chooses the path with lower cost, where the cost can be any parameter the network administrator defines. After that, it runs a RWA based on the first-fit approach. The RS is also capable of finding a disjoint lightpath for protection purposes. Both RMS and RS were implemented in Java, using Axis Java as SOAP engine and Apache Tomcat as web container.

The AS receives an username and password and verifies if this pair is valid. The LS stores any informations passed to it in a file, with the time that event occurred. In our case we are logging system exceptions and failure informations for statistical purposes. These services are also implemented in Java.

The optical network element was emulated using a modified Linux kernel. This kernel is Multiprotocol Label Switching (MPLS) capable and we used an MPLS label table entry to emulate an optical cross connection. Wavelengths are emulated using MPLS labels. We could use a range of labels to emulate a waveband switching and a whole network interface to emulate a fiber switching. For fiber switching emulation, using a packet filter (as provided by *iptables*) it

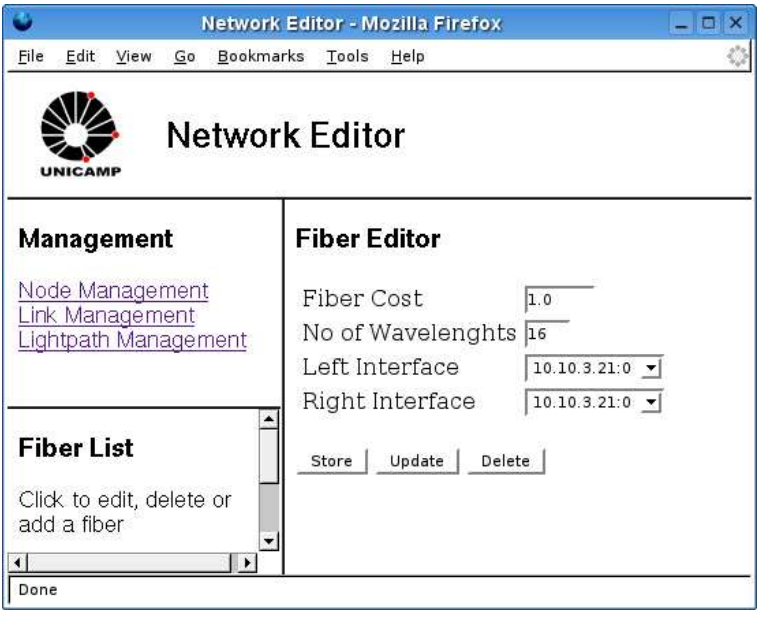

Fig. 3. Web interface of the Network Editor

is possible to redirect a packet arriving at a given network interface to another interface, no matter the packet contents. This is summarized in Fig. 4.

In our emulated scenario the Optical Switch Service must set the MPLS label table at the Linux kernel via system calls. As such, this service must be capable of performing low level actions at the kernel. In a real scenario, this service must run in the optical switch's internal processor (or in an adjunct processor such as a desktop). As a result, the service must be efficient and work with limited resources. Due to these reasons we decided to code this service in C++, as it is an efficient and compact object oriented programming language that still enables low level interactions.

Apache HTTP server and Axis C++ were used to develop this service. We decided to use a document literal wrapped style for all SOAP (Simple Object Access Protocol) bindings, as SOAP encoding is being not recommended for interoperability reasons. SOAP messages using document literal wrapped can be totally validated, as the body part does not contain any type encoding info and the message must be compliant with an agreed XML Schema [9, 10]. Moreover, the method name is present in the SOAP message, what simplifies the dispatching of the message at the server side to the right method implementation. Indications that the industry is abandoning SOAP encoding can be seen from its omission from the WS-I Basic Profile [11].

Using all these composed services the OCS is capable of establishing a connection inside the network. The interaction with the optical switches is accomplished concurrently, i.e., all the switches in the lightpath are cross connected almost at the same time. This can produce a good performance improvement

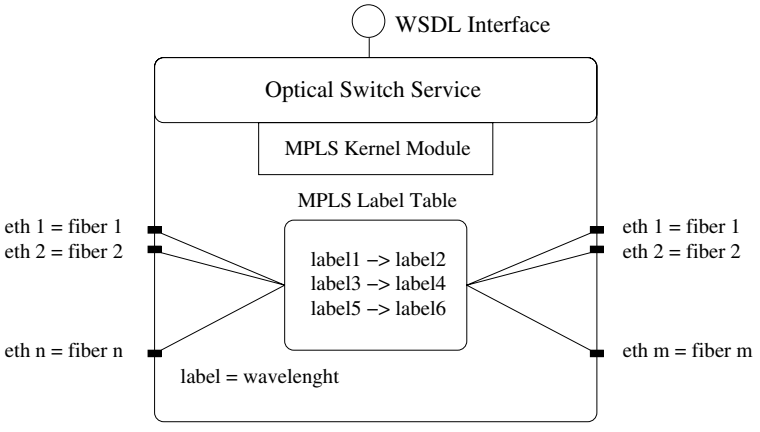

Fig. 4. Emulated optical switch

when compared to signaling protocols where the cross connection are performed sequentially.

When any link failure occurs or is cleared in the network the FMS must be notified. Currently we have implemented the *1:1* and *1:n* protection levels. On a failure the FMS acts on the ingress node, firstly dropping the failed connection and then adding the protection connection. The procedure is analogous when a fault is cleared. The FMS only acts on the ingress node because the rest of the lightpath is already set up.

Finally, all the traffic between the network client and the orchestration engine is transmitted over a secure channel, using Secure Socket Layer (SSL). This aims to protect user's authentication and exchanged data.

4.3 Tests and Evaluation

To test the implemented system we developed a command line client program to interact with the orchestrated service. The language used to code the client software is not relevant, since the SOAP messages exchanged with the service are XML Schema compliant. In our case we used Java language and Axis Java as SOAP engine. The following tests were performed to evaluate the OCS and FMS:

- Lightpath creation;
- Lightpath dropping;
- Fault restoration;
- Clear fault event.

In order to evaluate the performance impact of SOA in network services we focused on response times. As response time we have considered the time between a request message being sent and the response message arrival in client's machine. Further, in these tests we have not emulated the delay to set the cross connection

inside the optical elements and, as the cross connections are set concurrently, there was no significant difference when varying the number of nodes in the lightpath. The results of 100 measurements are shown in Table 1.

Table 1. Response time

Test	Response Time (*ms*)	Standard Deviation (*ms*)
Lightpath creation	122	13
Lightpath dropping	101	6
Fault restoration	125	5
Clear Fault	136	8

As optical connections generally have long duration the response time for lightpath creation is acceptable. It is important to remember that it includes the time of all processing needed to establish the connection, even those related to the business layer (e.g., logging). Depending on the kind of application, notably phone connections, the restoration response time can be very restrictive. The response time of the FMS for these kind of applications can be considered inadequate if compared to restoration times performed at Layer-2 (such as on Synchronous Optical Networks - SONET). The performance obtained by the FMS is adequate for applications with less stringent recovering requirements such as web browsing and video on demand.

The lightpath dropping and clear fault times are not important issues except in the case where the network is overloaded (all possible lightpaths are installed) and there is need for new connections. Even in these cases, the times obtained are very satisfactory.

The objective of our architecture is to provide quick and flexible development of new network services. With the aid of specialized tools and appropriated languages we consider we achieved our objective. The solutions are simple and robust, and the implementation validates completely the proposed architecture.

5 Related Work

An important related work is being developed under the Canarie [12] User Controlled LightPath (UCLP) Research Program [13], implemented and tested in the Canadian research network CA*Net4. UCLP allows end-users, either people or software applications, to treat network resources as software objects, provisioning and reconfiguring them within a single domain or across multiple, independently managed domains. This research explores new features and enhancements to the current implementation of UCLP through the use of Web Services workflow and orchestration to create "Articulated Private Networks". The main design features of this architecture are [14]:

– All network software, hardware, lightpaths and cross connects are exposed as Web Services;

– Web Services workflow are employed to build a universal control plane across instruments, lightpaths, cross connects, networks and software systems.

Different from UCLP approach, we have not allowed the client interact directly with the network elements inside a domain for security reasons. In our architecture a contractual interface for each administrative domain is exposed in order to provide services for the client, creating a layer over the services offered by the domain. This is more likely to happen, as network providers have serious restrictions in opening their networks for full signaling or management of network elements. Furthermore, using the recurrent service construction provided by SOA we can provide end-to-end services in a very structured way.

The work presented in [15] proposes Web Services as an embedded technology for home appliances. The work argues that this is not an unrealistic assumption as the price and capacity of embedded processors are becoming reasonable for this application. This current work proposes the use of Web Services inside network devices, based on the same arguments.

Similarly to the proposals presented in references [13] and [16] our architecture give the customer the ability to manage a set of service parameters considering customers' specific needs. Moreover, reference [16] gives the customers the ability to build dynamically their application using a service trader. This could improve the level of automation of our architecture and we are now considering a similar approach, but employing UDDI (Universal Description, Discovery and Integration) instead.

6 Closing Remarks

This paper proposes a service oriented architecture for deployment and management of services in connection oriented networks. This architecture brings to the network service provider a higher level of automation, integration and flexibility in the design, deployment, and management of network services. These activities are performed by orchestration scripts executing in standard orchestration engines. Two implementation instances of this architecture in the field of optical networks were developed in order to evaluate the feasibility of the proposed architecture.

One advantage of using composition to build services over an heterogeneous network is the elimination of interoperability bottlenecks. Instead of implementing complex and sometimes poorly standardized protocols, network equipment vendors can implement Web Service interfaces to control and manage their equipments. By publishing this interface, network operators and third party software vendors can control and manage network equipments by incorporating these interfaces into composition scripts. Contrary of network protocols, the Web Service interface need not to be fully standardized (they need only to expose similar functionalities).

As a future work we are considering the incorporation of policies into the orchestrated service in order to adapt the service use and management according to user privileges and profiles.

Finally, we believe that Web Services is a practical way to integrate different network domains. Network operators do not allow network signaling to cross their network boundaries due to stability and security reasons. The offering of inter-domain services via Web Services composition is more feasible and simpler as network providers have full control over the information exchanged in the inter-domain borders.

References

[1] Mike P. Papazoglou and Dimitris Georgakopoulos. Service-oriented computing: Introduction. *Communications of the ACM*, 46(10):24–28, October 2003.

[2] Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai, and Sanjiva Weerawarana. The next step in web services. *Communications of the ACM*, 46(10):29–34, 2003.

[3] World Wide Web Consortium (W3C). *Web Services Choreography Description Language Version 1.0*, December 2004. Working Draft.

[4] Keith Mantell. From UML to BPEL. http://www-128.ibm.com/developerworks/webservices/library/ws-uml2bpel/, September 2003.

[5] ActiveWebflow Professional. http://www.active-endpoints.com/.

[6] Oracle JDeveloper 10g. http://www.oracle.com/.

[7] BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems. *Business Process Execution Language for Web Services Version 1.1*, May 2003.

[8] ActiveBPEL. http://www.activebpel.org/.

[9] Russell Butek. Which style of WSDL should I use? http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/index.html, May 2005.

[10] Tim Ewald. The Argument Against SOAP Encoding. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/argsoape.asp, October 2002.

[11] Web Services Interoperability Organization. *Basic Profile Version 1.1*, August 2004. http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html.

[12] Canarie Inc. http://www.canarie.ca.

[13] Bill St. Arnaud, Andrew Bjerring, Omar cherkaoui, Raouf Boutaba, Martin Potts, and Wade Hong. Web Services Architecture for User Control and Management of Optical Internet Networks. *Proceedings of the IEEE*, 92(9):1490–1500, September 2004.

[14] Bill St. Arnaud. Web Services Workflow for Connecting Research Instruments and Sensors to Networks. http://www.canarie.ca/canet4/uclp/UCLP_Roadmap.doc, December 2004.

[15] Masahide Nakamura, Hiroshi Igaki, Haruaki Tamada, and Ken ichi Matsumoto. Implementing integrated services of networked home appliances using service oriented architecture. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 269–278, New York, NY, USA, 2004. ACM Press.

[16] Dirk Thissen. Flexible Service Provision Considering Specific Customer Needs. In *Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP'02)*, pages 253–260. IEEE, 2002.