

SOANet - A Service Oriented Architecture for Building Compositional 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, Sao Paulo, Brazil
 E-mail: {vsouza, eleri}@dca.fee.unicamp.br

Abstract—The high competition among network providers places a demand for new ways to design network services with near zero development time, low cost, and high degrees of customization and evolution. Customization allows to fit the service according to the customers' requirements, while evolution allows to adapt the service as soon as these requirements change. Moreover, customers are demanding the ability to manage the contracted services in order to keep service 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 development, customization, and customer-side manageability. The approach considers the network service as the composition (orchestration) of a set of basic services. A generalization of the architecture primarily aimed at connection oriented networks is introduced, extending it to overlay networks. Two services for the management of tunnels on optical networks are presented as a case study.

Index Terms—Service Oriented Computing, Service Oriented Architecture, Web Services, Orchestration, Choreography, Connection Oriented Networks, Overlay Networks

I. INTRODUCTION

THE high competitiveness, associated with the need of client fidelity and offer of customized services have driven network providers to consider aspects that were previously ignored in the development process of new network services. Firstly, the time-to-market of new services must be minimized. In other words, the time between the specification and the effective use of the service should be as small as possible. Moreover, network services should take into account specific clients' requirements such as aspects related to configuration, billing and quality of service. Customers are demanding the ability to manage certain service characteristics to take advantage of their business peculiarities, such as traffic patterns and security requirements. Obviously, price is always an important variable that should be reduced by reducing the

complexity of development, installation and management of services.

It is remarkable that network services present high diversity regarding the technologies employed by software entities at business, management and network levels. Network software entities are based on low level signaling protocols, such as RSVP-TE (Resource Reservation Protocol - Traffic Engineering) and UNI (User-to-Network Interface) from Optical Internetworking Forum [1]. The access to these protocols is made via operational interfaces, network management protocols such as SNMP (Simple Network Management Protocol) or specific proprietary APIs (Application Programming Interfaces). On the other hand, business software elements rely on high level software entities such as enterprise or Web components. Management is usually implemented with object oriented languages, being responsible for linking the business decisions and the network signaling protocols. This diversity is surely a complicating factor when network services featuring high level of automation and integration need to be designed, developed and deployed in a short period of time.

It is clear in this context that network providers need to devise a novel way of developing services. It is believed that service oriented architecture (SOA) is a reasonable solution for this problem [2], [3], [4] and the service composition mechanisms should play an important role in this scenario. A new *service* can be created by composing a primitive group of *services*. This recursive definition is important because it enables the construction of more complex services above a set of existent ones. For example, a Virtual Private Network (VPN) service may be constructed through the composition of services for connection management, authentication, fault management, and resource management. These services can be distributed in the service provider's network, some executing in the central office computers and others executing close or inside the switching equipments.

Furthermore, the service oriented computing (SOC) paradigm is an attractive solution since it offers a good degree of automation, integration, customization and flexibility in the creation, deployment and management of new services. In this work we propose a general service oriented architecture for the development, deployment and

This paper is based on "A Service Oriented Architecture for Deploying and Managing Network Services", by Victor A. S. M. de Souza, and Eleri Cardozo, which appeared in the Proceedings of the 3rd International Conference on Service Oriented Computing (ICSOC), Amsterdam, Netherlands, December 2005. © Springer-Verlag.

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

management of network services, focusing on connection oriented and overlay networks. The architecture, named SOANet (Service Oriented Architecture for Network Services), assumes that the basic services are Web Services and their compositions are governed by the orchestration and choreography of Web Services. The service creation consists on the edition of an orchestration script while the deployment phase consists simply on loading this script to a software engine appropriate for its execution. The interaction between services running on different network domains is regulated by a choreography contract. The edition of orchestration and choreography scripts shall be aided by specialized editing tools in order to improve productivity. Moreover, all the software infrastructure for communication with the basic and composed services can be generated automatically.

This article is organized as follows. Section 2 describes some related and recently published works. Section 3 presents the proposed architecture for development of network services. Section 4 describes the extensions to the former architecture [2] in order to provide the same benefits to the development of overlay networks. Section 5 presents the design of the two composed services developed for assessing the proposed architecture. Section 6 presents a qualitative and quantitative evaluation of the implemented systems. Finally, Section 7 closes the paper with some concluding remarks.

II. RELATED WORK

An important related work is being developed under the Canarie [5] User Controlled LightPath (UCLP) Research Program [6], 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.

Differently from UCLP approach, we have not allowed the client to 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 impose stringent 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.

Reference [4] describes an architecture for the creation of network services based on the composition of Web Services. In this work two types of network services were defined, atomic and composed services. The former exposes its functionalities to the end user of the network, implemented through the composition of atomic services. The central distinction with our work is that they use wrappers to interact with network elements. These wrappers translate the Web Services calls to proprietary commands (issued via SNMP or Telnet, for example). Our work extends the scope of Web Services to the network devices, without the utilization of wrappers or gateways. In this way, we obtain an architecture that enables the service composition in a more comprehensive and flexible way.

III. THE SOANET ARCHITECTURE

As the SOANet is a service oriented architecture, this section briefly exposes the definitions and concepts of SOA. In the sequence, the SOANet architecture is defined and a software development process suitable for it is described. The SOANet end-to-end service composition and the client-side framework are also presented.

A. Service Oriented Architecture

Service oriented computing is defined as "the computing paradigm that utilizes services as fundamental elements for developing applications" [7]. As the fundamental elements of SOC (services) have a high level of abstraction, the paradigm is suitable for developing and integrating large and complex systems. Legacy applications can even be wrapped as a service and be used in the new system. The integration between the basic elements is done through service interfaces, which must be well-formed (according to a certain agreed standard) and loose-coupled to the service implementation. In other words, the interface must exhibit only the external visible behavior of a service.

The SOC paradigm has gained the software industry's attention for its clear and modularized components, which can be easily aggregated and for providing a high level of software reuse. When compared to other distributed programming paradigms such as CORBA (Common Object Request Broker Architecture) and RMI (Remote Method Invocation), SOC presents several advantages, mainly, independence of platforms, well agreed standards, and a multitude of vendors.

One of the most important characteristics of SOA is that services can seamlessly interoperate, even running on different software platforms and implemented on different languages by each party. This is achieved through the use of a platform independent transport protocol and a neutral language for interface specification. Moreover, services can be composed to form another service with added semantic value, providing a flexible, low cost, and quick way of developing new applications. The services used in the composition (basic or elementary services through this

document) are modular and independent units of software, facilitating reusability and thus reducing the efforts and time required to construct new services.

Additionally, the SOA architecture offers a registration and search mechanisms in order to make the services location independent. Clients implement a lookup procedure for searching the registration directory for the suitable service. The client can specify several different constraints on the service being searched, as long as the service provider registers this information on the directory.

The request-response invocation style (original from the client-server model) is relaxed in SOA. This is a considerable advantage in the situation where long term transactions are held by the service provider, for it does not get a transport connection blocked waiting the service computation.

References [7], [8] present more detailed information on service oriented computing and service oriented architecture.

B. Overview of SOANet

The SOANet Architecture was originally described in [2]. The architecture was aimed at providing a quick and low cost way to design, develop and deploy novel network services for connection-oriented networks within one domain. As it is completely based on the service oriented architecture the SOANet features all SOA characteristics shown on Subsection III-A.

As stated before, novel network services need to be developed in a very short period of time and fit in the clients' requirements. The high level of code reusability present in SOA contributes to this requirement. Additionally, the coarse-grained granularity and loose coupling enables one to build arbitrarily complex systems more quickly. The basic assumption of SOA is that every software entity must be designed as a service. In order to apply this approach to the field of communication networks we logically classified the basic network functionalities into three levels:

- Business level: services related to the application business logic. This level includes service accounting, trading, dynamic negotiations, logging, billing, and subscription, and service level agreements (SLAs).
- Management level: functionalities related to the management of the network service. Software at this level performs functions such as accepting/rejecting requests from clients, applying domain specific policies, resource control and brokering, monitoring and configuring network equipments, and logging.
- Network level: functionalities related to the transport network itself. These software units provide facilities for configuration of network devices, monitoring of parameters, logging, route computation, and fault restoration. Additionally, these services might be able to invoke callback interfaces for notifying events such as alarms reporting failures.

At the present time, services at these three levels are implemented in different languages and platforms. Services at the business level are implemented with software components - Web components (*e.g.*, Java Server Pages, Active Server Pages) or enterprise components (*e.g.*, Enterprise Java Beans, Component Object Model). These services run on application servers. The management level software units are implemented with object-oriented languages such as C++ or Java and run on management workstations or desktops. In contrast, functionalities of the network level rely on low level signaling protocols such as RSVP-TE and SNMP or proprietary protocols and APIs. These services are typically implemented in the C programming language and run on embedded processors.

This vast heterogeneity of software entities makes the creation of a network service an arduous task. The integration of these components is commonly accomplished by the use of software gateways and adapters responsible for translating the decisions from one level to another. The use of gateways and adapters seriously compromise the time of software development and its evolution. They are dependent in nature to both sides they connect, and a minor change in one of the elements will demand a change in the gateway or adapter.

We claim that service oriented computing is a step forward toward the solution of these problems. Our fundamental assumption is that every software entity is implemented as a service available in the network. In order to develop a new service, one must specify and implement the basic services that are needed by this new service. In order to ease the development of the system as a whole, the basic services have their interactions governed by service composition mechanisms. The process of creating a new service is then a matter of composing the basic services in an orchestration script. This phase can be helped by the utilization of development tools, in which the orchestration script is created by simple *drag-and-drop* operations.

The basic architecture is presented in Fig. 1. The services are logically grouped accordingly to the classification previously discussed.

One of the features of paramount importance to the architecture is associated to the fact that each network element (*e.g.*, network router, switch) is enhanced with a service interface. The management, control, and monitoring of network elements are completely handled by the execution of the appropriate method in this interface. The first consequence of this assumption is that we can eliminate the need for signaling protocols. By coordinating the management functions in the network element service interface it is possible to establish connections and reserve resources, among others. Basically, all functionalities provided by a signaling protocol can be implemented through the composition of services available at the network elements.

The introduction of a service interface in the network elements does not represent a challenge for the vendors since today most of the network elements already hosts

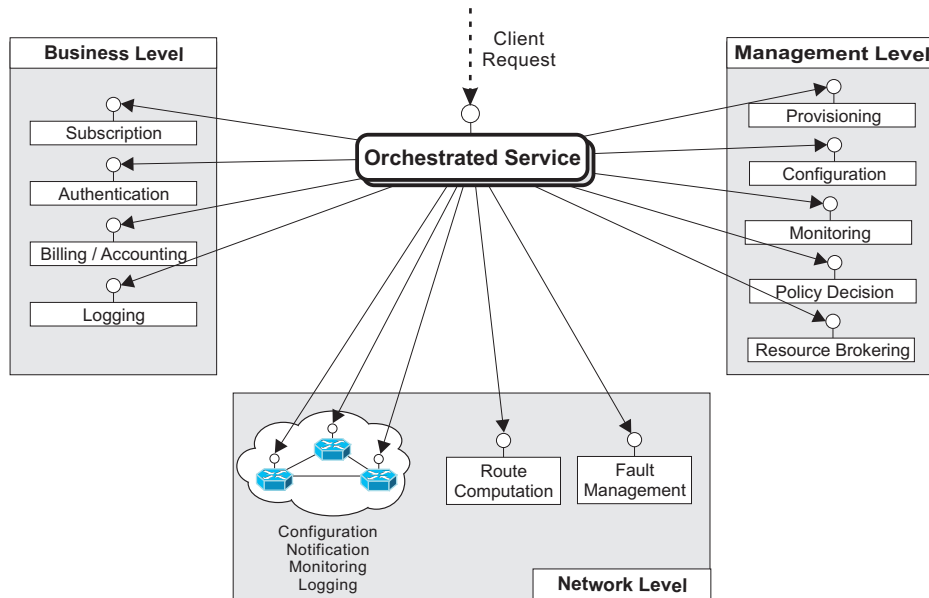


Fig. 1. A service oriented architecture for developing network services.

a Web server for enabling Web-based management. Extending this server to provide service-oriented capabilities is a minor step, without impact on the development of the embedded software or on the final cost of the equipment.

Therefore, SOANet gives the system designer the option not to use a signaling protocol. Of course, the protocols embedded in the equipment can be used, given the condition that all equipments are compatible, *i.e.*, implement the same protocol. Even in this case, the access to the node initiating the signaling is done via service interface. In the situation where the network is heterogeneous and different vendors implement protocols unable to coexist, the management via service interface represents a reasonable solution. Furthermore, the need for gateways is eliminated, bringing the management, business and network levels closer.

C. Service Creation

Once the basic services are specified, designed, and implemented, the creation of a new service is a matter of composing these services in an orchestrated way. The existing software development cycles can be used to implement a new service as exemplified below.

- 1) Requirements analysis. The requirements of the new service to be developed should be identified and described in a requirement analysis document. This document should contain all functional and non-functional service requirements.
- 2) Initial services specification. The requirements of the new service should be classified functionally and mapped to a set of basic services. In the end of this phase, one should have a list of the basic services and its respective functional and non-functional requirements.

- 3) Class and package diagrams. Based on the specified basic and compositional services, a class or package diagram should be constructed. Only the service interfaces are specified, along with data exchanged in each service function. The class diagrams for each service should not be shown in this diagram, as it depends on the particular design of the service.
- 4) Identification of existent services. A list the already available services is elaborated. The services that are totally compliant to the class diagrams previously constructed should be straightly reused. Legacy software should be wrapped by a service interface.
- 5) Implementation of elementary services. Services that are not available in the network should be implemented from scratch or using legacy software, when applicable. An alternative to implementation is to "rent" the service from a service provider offering a similar service in its network.
- 6) Implementation of compositional service: The compositional service should be implemented using an orchestration language. Graphical development tools may be utilized to ease the orchestration script creation process.

A service/network provider implementing the architecture for a short period would probably have some basic services that can be reused, as there are functionalities that are very common on every network service (*e.g.*, billing). As the time using the architecture grows, much more services are available for further reuse. It can be noted that the service oriented architecture provides great flexibility on the service creation process. This is a desired characteristic, since it may significantly decrease the cost of service development.

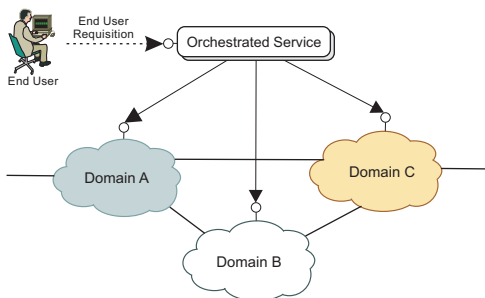


Fig. 2. End-to-end service composition.

D. End-to-end Service Composition

The SOANet architecture is applicable to one network domain. Nevertheless, using the concept of recursive service composition it is possible to create end-to-end services from the services supplied by each domain. Fig. 2 illustrates this idea.

Solutions that employ signaling between domains are not well accepted by network providers. Opening the network for external signaling has never really occurred due to security and privacy issues. Network managers resist to open their equipments to external management, for example, by allowing the establishment of source routed circuits. The solution presented in this work solves these issues since the equipments on a domain are managed locally. Based on internal policies the external resource allocation requisitions can be accepted or rejected. The end-to-end service may be implemented by one network provider or even by a third party (e.g., a content provider). In both cases, the agent implementing the end-to-end service must have contracts with all the service providers of each covered domain in order to offer a service with aggregated value. It is also possible that each domain implements its own orchestrated service in order to provide end-to-end services to their clients. In this approach the provisioning of end-to-end services is completely distributed, resulting on better scalability of the overall system.

E. Client-side Architecture

The client of the services sends its requisition through the interface exhibited in Fig. 1. Only the external behavior of the service is represented on this interface. The client must obtain the specification of the interface it wishes to interact with and build its requisition accordingly.

The possible clients of the service exposed by one network domain are a final user, a network provider (composing end-to-end services), or a service provider. A final user will demand for connections with a given machine that is directly connected to the domain it is interacting with. On the other hand, the network and the service provider own a business process, which is a software utilizing services of other domains to achieve its objectives. In the later case, it is mandatory that a well defined contract be used to unambiguously specify the

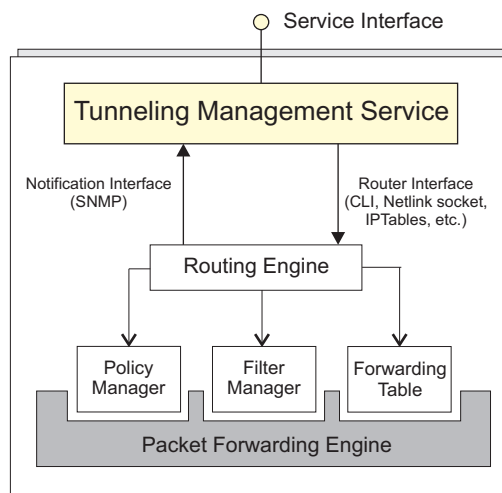


Fig. 3. Example of an enhanced network equipment fitted with a service interface.

interaction between services in distinct domains. Therefore, a choreography script may play a very important role here, since it defines formally the interaction between the service consumer and provider. As a result, in SOANet the interaction between services of different domains always rely on a choreography contract.

IV. THE EXTENDED SOANET ARCHITECTURE

The Extended SOANet architecture is a general framework for the development of network services. Originally designed for connection oriented networks, the architecture was extended to overlay networks by generalizing the concept of a connection. In data networks, a connection is a communication channel (or circuit) established along a path from a origin to a destination. The network elements (routers, optical cross-connects, etc.) in the path are responsible to establish and maintain connections with the aid of signaling protocols.

The concept of *connection* present in SOANet was generalized by introducing the concept of *tunnel*. Tunnel is a more abstract concept when compared with network connection in the following senses:

- tunnels can perform extra functions over the transported data (e.g., cryptography and authentication);
- tunnels can be established on networks that do not support connections (e.g., IP networks);
- tunnels can be aggregated into tunnel trunks for simplifying their management.

Our definition for a *tunnel* is “a transparent channel of communication established between two network elements, possibly crossing domain borders”. A tunnel can be supported by a lightpath in case of optical networks, a Label Switched Path (LSP) in case of IP networks, a time-division channel in synchronous networks such as SONET (Synchronous Optical Network), and so on. Fig. 3 illustrates one example of a network element, enhanced by the service for tunneling management.

The Tunnel Management Service (TMS) must be installed in every network/overlay node. Essentially, it exposes monitoring, management, and control of routing (or switching) functions of the network equipment. It interacts directly with the routing core, sending commands and receiving notifications. This is accomplished via points of interaction that are already present in network elements (such as SNMP). The router core is then responsible for setting up filters, policies, and forwarding tables that will impact over the packet forwarding engine.

Several functionalities can be exposed in the tunneling management interface. Examples of such functionalities include life cycle functions (*e.g.*, creation, destruction, and rerouting of tunnels), tunnel configuration functions (*e.g.*, security functions), quality of service assessment (*e.g.*, delay, jitter, and packet loss for a given tunnel), and statistics gathering functions (*e.g.*, available bandwidth and traffic parameters). These functions may be employed by a traffic engineering entity performing optimization of the overlay network. Examples of optimizations include load balancing among tunnels, traffic swapping among tunnels, and rerouting of tunnels.

By exposing a high level interface, a network can support multiple overlay networks at the same time, each one employing its own topology, tunneling scheme, and control protocols such as routing and management protocols.

V. IMPLEMENTATION DESCRIPTION

In order to validate and assess the proposed architecture in one network domain we have designed and implemented two composed services for managing tunnels on GMPLS (Generalized Multiprotocol Label Switching) networks. Tunnels are established in order to create overlay networks above an GMPLS network. Example of overlay networks include virtual private networks (VPNs), content-distribution networks, and mobility-aware networks. The first implemented composed service, named Tunnel Provisioning Service (TPS), aims to establish and uninstall tunnels on GMPLS networks. The second service, Fault Management Service (FMS) is aimed at restoring and clearing network faults.

A. Requirement Analysis

The Tunnel Provisioning Service must offer facilities for creation and elimination of primary tunnels. One protection tunnel may be associated to this primary tunnel. The protection tunnel may be dedicated (protecting only one primary tunnel) or shared (protecting more than one primary tunnel). The primary and protection tunnels must be disjoint. The client requires installation of a tunnel by supplying its starting and ending points. Optionally, the client may supply the route and tunnel constraints such as traffic parameters, protection parameters, and physical parameters (wavelengths, channels, etc.). The client of the service must be billed for the service usage according to a contract previously agreed (SLA). All requisitions issued

should be compatible with this contract. The service must authenticate the network client before processing the requisition.

The Fault Management Service must offer facilities for restoration and clearance of faults. Restoration is the process of switching the traffic from primary to secondary tunnels when a fault affects the primary tunnel. Clearance is the reverse process taking place when the fault no longer exists. The service must be capable of restoring dedicated and shared tunnels. In both services, when errors occur during the service execution, details of the errors must be stored in a persistent manner for further processing.

The most important non-functional requirement to both services is the security and data privacy. It is mandatory that the client supplies a set of credentials for the service, which in turn must be checked by the system. The requisitions and authentication data must be sent in a secure way to the service provider, ensuring data privacy.

The time needed by the service for installation of a tunnel and for restoration of a fault cannot be excessive outsized. Network operators agree that the restoration should be realized within 50 milliseconds. However, for the proposed system an acceptable value would be 500 milliseconds, due to the fact that this restoration is implemented at the network's application layer. For the tunnel installation we consider that one second is a reasonable value, since tunnels remain installed for long periods of time.

Both services must be highly stable and robust. They must feature high availability and short downtime periods.

B. Basic Services Specification

The requirements of TPS and FMS were functionally classified and mapped to a set of basic services. Of course, this classification is not unique, being dependent on decisions taken by the service designer. If a basic service is well designed it is very likely that it will be reused in other service compositions. The identified elementary services were:

- Tunneling Management Service (TMS): service exposed by the network elements providing switching facilities in the network.
- Resource Management Service (RMS): service responsible for keeping the network state updated, managing all data associated to installed tunnels, currently available resources, and SLAs. Also, it keeps information about the protection tunnels bound to a given primary path.
- Routing Service (RS): service responsible for the route computation in the network, given one ingress node, one egress node and the available network resources. It must be capable of allocating resources such as bandwidth and wavelengths along this route. Furthermore, it is responsible for finding a disjoint route for protection tunnels.
- Authentication Service (AS): service responsible for authenticating the users of the network service.

- **Logging Service (LS):** service responsible for keeping information regarding errors while processing the composed service persistently.
- **Billing Service (BS):** service responsible for accounting the resources reserved and effectively used by a client and generate a bill based on this information.

Note that all functional requirements of the composed services were mapped to one of the basic services. Non-functional requirements such as stability and robustness shall be considered in the implementation phase.

C. Class Diagrams

Class diagrams model the dependences among the services. An example of such diagram is presented in Fig. 4, the class diagram for the Tunnel Provisioning Service. The composed service is represented by the service name added of the suffix *Engine* to show that, in fact, the calls to the basic services are made by the orchestration engine (fed with the orchestration script). The parameters of each method were omitted.

This class diagram was obtained based on the functional requirements previously discussed. Each one of the specified services had its functionalities directly exposed in a service interface. The interfaces were named by adding the the suffix *PortType* to the service name. The data that must be kept in a persistent way are managed by beans that interface the system and a database according to a three-tier architecture (classes of interface, control and persistence). The control classes effectively implement the logic exposed in the interface classes and make calls to the persistence classes when needed. This can be cleared visualized in the Fig. 5, which depicts the class diagram for the Resource Management Service.

TPS and FMS were not implemented simultaneously. When the FMS was designed the TPS was already on a testing phase. This was deliberated in order to evaluate qualitatively code reuse in the FMS implementation. The results were very satisfactory for all the basic services needed to implement the FMS were reused from the set of available services, without any modifications.

D. Implementation of Elementary Services

All the basic services as previously described were implemented from scratch. The architectural decisions for the implementation, as well as the details of the chosen platforms and languages will be presented in the sequence.

The Routing Service (RS) is responsible for finding a tunnel from a source node to a destination node within the domain. RS needs to know the topology of the entire network domain and currently reserved resources. This information is obtained from the database maintained by the Resource Management Service. The service implements a Shortest Path First (SFP) procedure based on the cost of each link, where the cost can be defined by the network administrator. In a next step, resources are allocated to the path on a first-fit basis, implementing a Route and Wavelength Assignment (RWA) algorithm [9].

The Resource Management Service (RMS) is responsible for keeping the network state updated. The topology and the current state of the network are maintained in a database. When any resources are allocated or released in the domain the RMS should be notified in order to keep the database in a consistent state. The physical network topology, including nodes, link, costs, and resources are inserted in the database by the system administrator with the aid of a Web interface.

A charging model was designed and implemented in the Billing Service. In this model the client is initially charged for the setup of the primary and protection tunnels. Another component of the charging model is dependent on the time the given resource was allocated for that client (even if it is not being used). Other variables that can be taken into account is setup and holding priorities if preemption of resources is considered (a rare situation in practice).

The Authentication Service is a very simple service that just checks the supplied username and password against a database. The Logging Service stores any information received (sequence of characters) in a logging file, together with the time of occurrence of that event, for further processing purposes.

All services, except TMS, were implemented using the Java programming language, Axis Java as a SOAP (Simple Object Access Protocol) engine and Apache Tomcat as Web container. The MySQL database engine was employed for persistent storage.

The Tunnel Management Service (TMS) was installed in each switching equipments of the domain in order to provide high level tunneling functionalities in the equipment. Since in this case study we are considering optical networks, tunnels are established through optical cross connections. Optical switches were emulated by MPLS-capable (Multiprotocol Label Switching) Linux routers. MPLS allows the labeling of IP packets speeding up forwarding and allowing traffic engineering such as constraint-based routing. Optical wavelengths were emulated by MPLS labels and whole fibers were emulated by Ethernet links.

In this emulated scenario, TMS must alter the MPLS label table in the Linux kernel via system calls. Thus, the service must be capable of low level interactions with the kernel. In a real scenario, this service must execute inside optical switches, being efficient and capable of running with limited amount of memory and processing power. Therefore, the C++ programming language was used to implement this service, since it is an efficient object oriented language that permits low level interaction. The Apache HTTP Server and the SOAP engine Axis C++ were used for the service execution and creation. The service requires the very basic features of these tools (SOAP message processing and dispatching).

E. Implementation of Compositional Services

With all the basic services implemented we started the implementation of the composed services. Several possi-

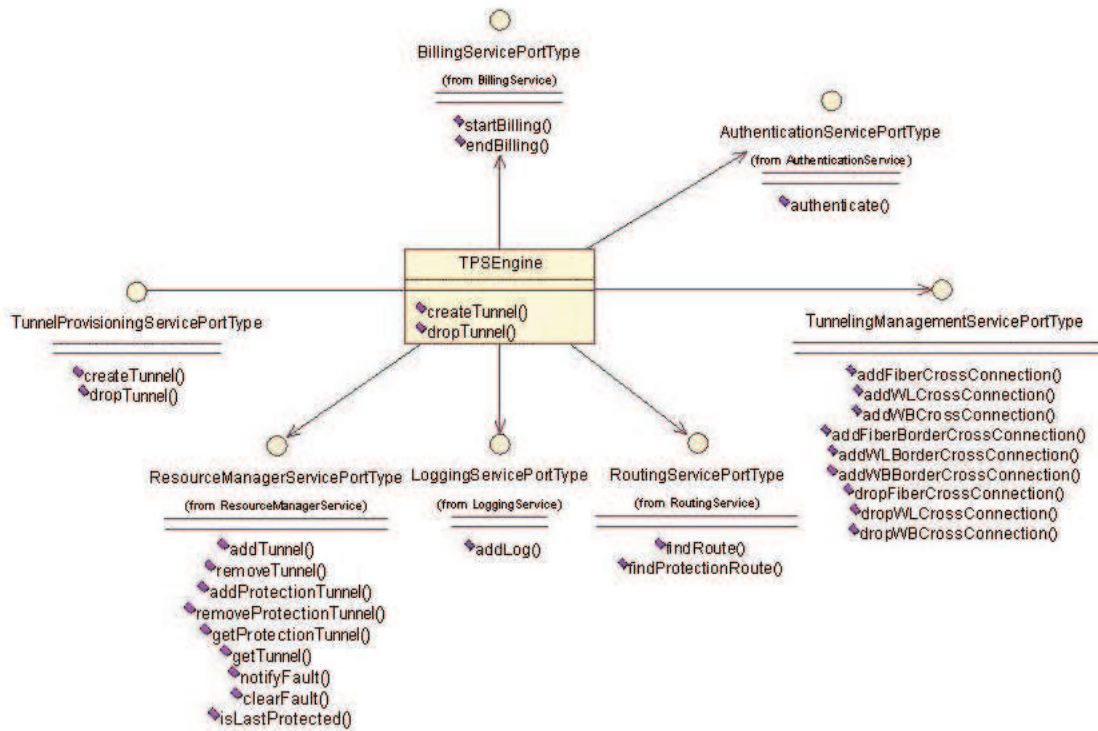


Fig. 4. UML class diagram for the Tunnel Provisioning Service.

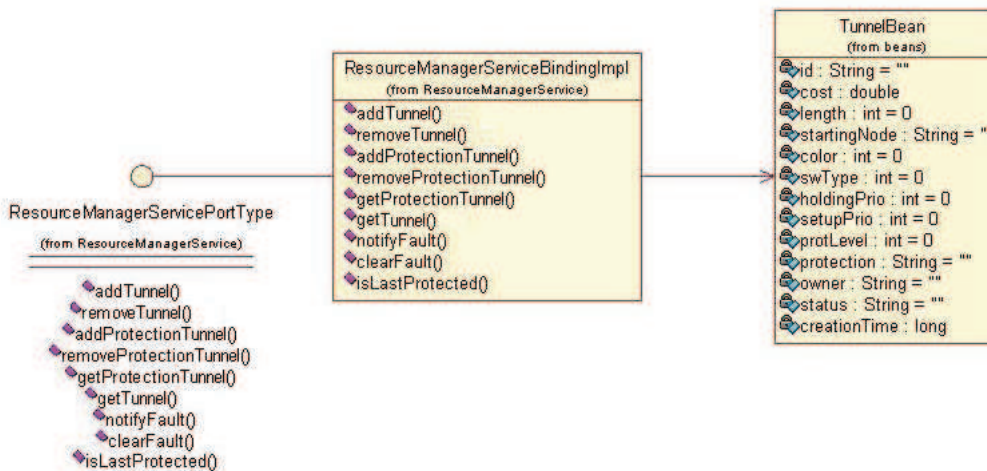


Fig. 5. UML class diagram for the Resource Management Service.

bilities arose when choosing the orchestration language to be used. We chose to use BPEL [10] (Business Process Execution Language for Web Services), since it is the mostly spread out and mature standardized orchestration language. In addition, it is the language which more native support provides to common workflow patterns [11].

The orchestration engine elected for use was ActiveBPEL [12] for being open source and totally compliant to the BPEL 1.1 specification. The tool for development of the orchestration scripts was ActiveWebFlow Professional [13], an Eclipse plug-in. This development interface is integrated with the orchestration engine,

easing the service deployment phase. It also provides means for depuration of errors. A Web interface shows statistical data about the orchestration server, such as number of active processes, number of installed services, processes execution state, among others. The process of development, depuration and deployment of the service is completely supported by the development interface.

Using the compensation mechanism provided by the BPEL language we are able to support rollback in tunnel 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 tunnel 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 earlier taken. The orchestration engine is in charge of coordinating the compensation actions that must be taken (cross connection releases in this case).

The interaction between the composed services and the optical cross connects is accomplished concurrently, *i.e.*, the cross connects in the tunnel are ideally switched at the same time. This technique can yield a considerable performance improvement when compared to signaling protocols where the cross connections are performed sequentially.

For privacy and security reasons, all messages sent between the network client and the orchestration engine are transmitted over a secure transport channel using Secure Socket Layer (SSL). As a consequence, the client's username and password provided in each requisition are not subject to naive threats. The document literal wrapped style of SOAP bindings [14] was used in all services according to WS-I Basic Profile [15], a practical guide for the construction of interoperable Web Services.

VI. EVALUATION

A. Tests

In order to test the developed services four command line client programs were implemented to interact with the orchestrated services. The programming language used to code the client is not relevant, given that the SOAP message exchanged with the service is compliant with the XSD¹ schema specified. In the case, the Java programming language and the Axis Java SOAP API were utilized. The tests conducted covered the functionalities of tunnel creation, tunnel elimination, fault restoration, and fault clearance.

To evaluate the impact yielded by the use of a SOA architecture in the network services field, focus was given to the response times. The response time was considered as the difference between the moment in which the message was sent by the client and the moment in which the answering message arrived in the client's machine. In these tests the delay for setting up a cross connection in the optical element was not emulated, *i.e.*, the emulated optical switch answers a requisition much faster than a real one. There is no significant difference on the tunnel setup time in function of the number of nodes in that tunnel, because the cross connections are set up concurrently.

The command line client programs were installed in one machine and the orchestrated services, the BS, LS, AS, RMS, and RS in another machine. The TMS was installed in each machine emulating the optical switch. All these machines were located in the same local area network.

All the tests were conducted in a controlled environment, a local area network featuring ideal conditions (*e.g.*, low traffic). For being realized in a local network, there is no considerable communication delay between the client and the orchestration server. Therefore, the results obtained should be considered optimistic. The results of 100 measurements are exhibited in Table I.

As optical connections generally have long duration the response time for tunnel creation is acceptable. It is important to emphasize that the response time herein shown includes all the processing needed to establish a tunnel, even those one related to the business level.

Depending on the type of application, namely voice connections, the restoration time may be very restrictive. The response time of the FMS for this kind of applications might be considered inadequate when compared to the response times obtained in Layer-2 restoration mechanisms (as in SONET networks). The performance of FMS is adequate for applications with requirements more tolerant to the restoration time, for example Internet surfing or video-on-demand.

The response time for tunnel elimination and fault clearing do not represent impact in the system performance, except in the case where the network is overloaded (all the possible tunnels are installed) and there is need for the installation of new ones. Even in these uncommon and restraining scenarios the results for both were very suitable.

In a general way, the response times obtained in the tests were satisfactory, showing that the impact of utilization of a SOANet in the network services field is not relevant.

The major objective of the architecture is to provide means for quick and flexible development of new network services, providing customization and evolution of the service. Both services were developed in a short period of time, with the aid of development tools for service compositions. The FMS did not demand any new elementary service, reusing only the set of available services in the network (implemented earlier for TPS). The flexibility obtained in the composition is remarkable and is essential to provide a good level of customization in the development of new services.

B. Comparative Analysis

The SOANet architecture proposes a new paradigm for the development of network services. It is a novel proposal in the sense that it allows the replacement of traditional network protocols by higher level services. However, no matter their interoperability and integration problems, these protocols are widely accepted and deployed throughout the service providers' networks.

The most innovative characteristic of the architecture is the exposition of management, control and monitoring functionalities of each network element as a service. Doing so, an orchestration script may substitute signaling protocols previously used, for example, for establishment of connections. Signaling protocols distribute the system

¹XML Schema Definition

TABLE I
RESPONSE TIMES IN TUNNEL MANAGEMENT.

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

intelligence in the sense that each node is capable of processing signaling messages and taking decisions based on them. On the other hand, the SOANet architecture has a certain degree of centralization as one orchestrated service is responsible for providing services to the client within one domain.

The existence of one unique orchestrated service per domain may be considered a single point of failure and bring poor performance when a high number of requisitions is received simultaneously. However, the architecture can be relaxed to accommodate more than one orchestrated service per domain (replication of the service). In this approach, scalability, reliability, and performance issues are strongly reduced. Each border network element could expose orchestrated services to the clients. Obviously, by using distributed compositional services it is necessary to control the access to shared resources (and services) to maintain a consistent state.

The option of use of a signaling protocol wrapped by a Web Service makes the architecture suitable for a widest range of network operators. Only a few network elements may be enhanced by the addition of a service interface, offering a good way to communicate management decisions down to the equipments.

On the other hand, the centralization of the orchestration service bears advantages. One of them is the possibility of concurrent installation of connections in each network element. Generally, signaling protocols install connections sequentially, *i.e.*, each element only installs the connection after the previous element have already installed. This fact might generate a significant reduction in the establishment time of connections, mainly when the number of network elements in a tunnel is high (say, more than 10). In this case, the SOANet architecture does not suffer any degradation, resulting in a constant compartment ($O(1)$), while traditional signaling protocols result in a linear compartment ($O(n)$).

VII. CONCLUSION

This paper proposes SOANet, an extensible architecture for development, deployment, and management of services in communication networks based on the service oriented architecture. 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. A prototype of the architecture in the field of optical networks was developed in order to assess the feasibility of the proposed architecture.

One additional advantage of using composition to build services over heterogeneous networks 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 borders due to interoperability and protection 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.

ACKNOWLEDGMENT

The authors would like to thank Ericsson Brazil for its support. We thank the Ph.D. student Tomas Badan for contributing with his MPLS Kernel implementation, and Professor Mauricio Ferreira Magalhaes for its contribution to the architecture design.

REFERENCES

- [1] "User-to-Network Interface," Optical Internetworking Forum, <http://www.oiforum.com/>.
- [2] V. A. S. M. de Souza and E. Cardozo, "A Service Oriented Architecture for Deploying and Managing Network Services," in *Proceedings of the 3rd International Conference on Service Oriented Computing*, F. C. e. P. T. B. Benatallah, Ed. Springer-Verlag, December 2005, pp. 465–477.
- [3] V. A. S. M. de Souza, "A Service Oriented Architecture for Developing, Managing and Deploying Network Services," Master's thesis, State University of Campinas, Campinas, February 2006.
- [4] N. Nishiyama, K. Nishikawa, F. Ito, and Y. Suzuki, "Composing User Network Operation Services Using Web Service Composition Techniques," in *Second IEEE Consumer Communications and Networking Conference (CCNC 2005)*. IEEE, Janeiro 2005, pp. 139–143.
- [5] "Canarie Inc." <http://www.canarie.ca>.
- [6] B. S. Arnaud, A. Bjerring, O. cherkaoui, R. Boutaba, M. Potts, and W. Hong, "Web Services Architecture for User Control and Management of Optical Internet Networks," *Proceedings of the IEEE*, vol. 92, no. 9, pp. 1490–1500, September 2004.
- [7] M. P. Papazoglou and D. Georgakopoulos, "Service-oriented computing: Introduction," *Communications of the ACM*, vol. 46, no. 10, pp. 24–28, October 2003.

- [8] M. P. Papazoglou, "Service-Oriented Computing: Concepts, Characteristics and Directions," in *WISE 2003: Proceedings of the Fourth International Conference on Web Information Systems Engineering*. IEEE, December 2003, pp. 3–12.
- [9] H. Zang, J. P. Jue, and B. Mukherjee, "A Review of Routing and Wavelength Assignment Approaches for Wavelength-Routed Optical WDM Networks," *Optical Networks Magazine*, vol. 1, January 2000.
- [10] *Business Process Execution Language for Web Services Version 1.1*, BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems, May 2003.
- [11] W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede, "Web Services Composition Languages: Old Wine in New Bottles?" in *EUROMICRO'03: Proceedings of the 29th EUROMICRO Conference New Waves in System Architecture*. IEEE, Setembro 2003, pp. 298–305.
- [12] "ActiveBPEL," Active Endpoints, <http://www.activebpel.org/>.
- [13] "ActiveWebflow Professional," Active Endpoints, <http://www.active-endpoints.com/>.
- [14] T. Ewald, "The Argument Against SOAP Encoding," October 2002, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsoap/html/argsoape.asp>.
- [15] *Basic Profile Version 1.1*, Web Services Interoperability Organization, August 2004, <http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html>.

Victor A. S. M. de Souza received the B.S. degree in Computer Engineering (2004) and the M.S. degree in Electrical Engineering (2006) from State University of Campinas, Brazil. Currently he is a Ph.D. student at State University of Campinas holding part of his researches at Ericsson Networking Labs in Stockholm, Sweden. His research interests are peer-to-peer systems, network management, distributed systems, and security.

Eleri Cardozo received the B.S. degree from University of São Paulo, Brazil (1978), the M.S. degree from Technological Institute of Aeronautics, Brazil (1981), and the Ph.D. degree from Carnegie Mellon University, USA (1987). Currently he is a Professor of Electrical and Computer Engineering at the School of Electrical and Computer Engineering, State University of Campinas, Brazil. His research interests include distributed systems, computer networks, and software engineering.