Design and Implementation of Web Services-based NGOSS Technology Specific Architecture

Mi-Jung Choi[1], Hong-Taek Ju[2], James W. Hong[1], and Dong-Sik Yun[3]
[1] {mjchoi, jwkhong}@ postech.ac.kr, *Dept. of Computer Science and Engineering, POSTECH, Korea*
[2] juht@kmu.ac.kr, *Dept. of Computer Science, Keimyoung University, Korea*
[3] dsyun@kt.co.kr, *KT Network Technologies Labs, KT, Korea*

To cope with frequent changes and functional additions of operation and support systems (OSSs), a guideline of OSS's architecture and development methods is needed. TeleManagement Forum (TMF) has provided NGOSS technology-neutral architecture (TNA), which describes major concepts and architectural details of the NGOSS architecture in a technologically neutral manner. The TNA can be mapped onto technology-specific architectures (TSAs) using specific technologies such as XML, Java and CORBA. Web Service, a distributed and service-oriented computing technology, can be also applied to NGOSS TSA. In this paper, we provide a design and implementation of Web Services-based TSA in accordance with the architectural principles of TNA and the performance evaluation of our proposed system. Our work can be used as a guideline for anyone planning to develop a Web Services-based NGOSS TSA.

Keywords: *NGOSS, TNA, TSA, CCV, Framework Services, Contract, Web Services*

I. Introduction

The number and complexity of communication services have exploded in the last few years and this trend will likely continue in the near future. Due to a rapid evolution of telecommunication technologies, products and services are also multiplying. Moreover, as the data communications, telecommunications, and other forms of communication converge, the complexity and size of networks that support the growing number and range of services is quickly increasing. The rapid growth of the Internet and the convergence of communication networks have become the main drivers for a flexible and scalable operations support solution for today's telcos' services and systems [2]. For this reason, TeleManagement Forum (TMF) [1] has proposed a Next Generation Operations Systems and Software (NGOSS) [4] framework to improve the management and operation of information and communication services. The goal of NGOSS is to facilitate the rapid development of flexible, and low cost ownership, as well as Operations and Business Support Systems (OSS/BSS) solutions to meet the business needs.
The service proliferation requires that new generation OSS/BSS system architectures must be capable of fulfilling a whole new set of unprecedented requirements that will arise due to the rapid service development. Irrespective of applications, platforms or technologies, OSS architectures must be able to:

- Define service components with various attributes
- Support short service development cycles

- Avoid service introduction delay, i.e., rapid launch of new services
- Deliver new services or service bundles in a scalable, repeatable manner without massive duplication of effort
- Automate configuration management to facilitate faster installation rates

The TMF's NGOSS technology-neutral architecture (TNA) [5], which is sustainable through technology changes, satisfies these requirements. NGOSS TNA describes major concepts and architectural details of the NGOSS architecture in a technologically neutral manner and emphasizes a service-oriented approach based on integration of well-defined collaboration contracts. It also targets the use of commercial off-the-shelf (COTS) information technologies, instead of technologies unique to the telecommunications industry, as many legacy management systems have done in the past. This approach significantly reduces costs and improves software reuse and operational flexibility, enabling NGOSS-based systems to support a range of new services and new technology environments more easily.

The TNA can be mapped to specific technologies for the purpose of implementing and deploying an NGOSS system. The separation of technology-neutral and technology-specific architectures (TSAs) enables OSS developers to choose the 'best fit' management components and technologies for their management capability. TMF has proposed three technology application notes that describe the mapping of TNA onto specific technologies such as XML [6], CORBA [7], and Java [8]. However, these application notes do not explain detailed application methods of technologies to the TNA components or any implementation guidelines. Web Service [11], an emerging technology with the powerful support of standardization, is a promising solution for NGOSS TNA. Web Service is a distributed and service-oriented computing technology with strong support that is freely available from the industry. Therefore, it satisfies the service-oriented architecture (SOA) of NGOSS and supports COTS tools that can be applied to NGOSS TNA components [3]. The adaptation of the SOA to create an environment, where there are granular and loosely coupled OSS components with standardized interfaces, can plug and play over a common infrastructure [2]. In this paper, we summarize the state-of-the-art NGOSS architecture. We propose a technology-specific architecture using Web Services in accordance with TNA principles [5], and provide detailed application methods and implementation details using Web Services technologies. We also present performance evaluation results and determine a bottleneck component in our Web Services-based system. We hope that our work can be used as a guideline for anyone planning to develop a Web Services-based NGOSS TSA.

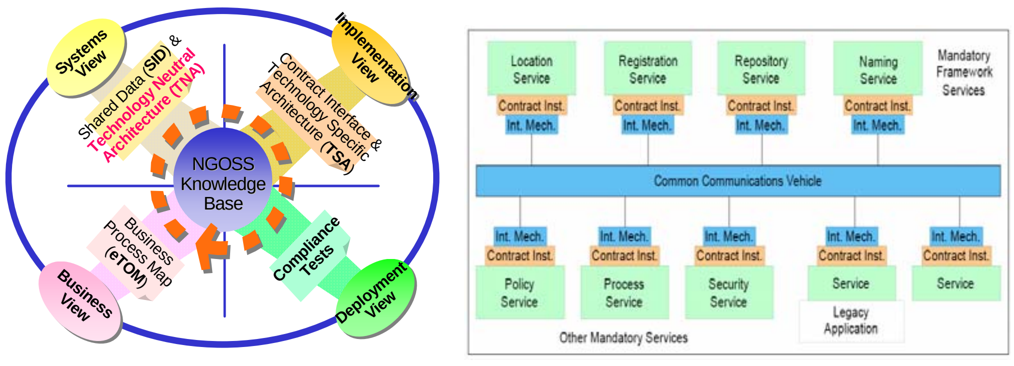
The organization of this paper is as follows. In Section II, we give an overview of NGOSS architecture with NGOSS TNA and TSAs. Section II also investigates three technology application notes. In Section III, we examine the alignments with NGOSS TNA using Web Services technologies, and describe our design of Web Services-based TSA. In Section IV, we demonstrate a case study. In Section V, implementation details of Web Services-based TSA in accordance with the case study are given. Section VI shows performance evaluation results of our system and shares our experience for implementing a Web Services-based NGOSS system. Finally, we conclude our work and discuss possible future work in Section VII.

II. Overview of NGOSS Architecture

In this section, we first briefly describe an overview of NGOSS and NGOSS TNA. Next, we summarize three technology application notes of TSA proposed by TMF and OSS/J initiative.

II.1. NGOSS Framework

NGOSS [4] is a comprehensive and integrated framework for developing, procuring, and deploying operational and business support systems and software that enables service providers and their suppliers to automate business processes and have the agility to respond to ever changing customer needs and market landscapes. TMF has developed NGOSS as a toolkit of industry-agreed specifications and guidelines that cover critical business and technical areas. Figure 1 (a) illustrates the NGOSS framework and four views [4]. Conceptually, there are four quadrants, each representing the aspects of defining, designing, implementing and operating an NGOSS solution which are termed as Business, System, Implementation, and Deployment views, respectively. A key advantage of this approach is that it provides trace-ability from business definition of the solution through its architecture, implementation and deployment specification.



(a) NGOSS Framework (b) NGOSS TNA

Figure 1. NGOSS Framework and TNA

In this paper, we focus more on the System and Implementation views. The System view is primarily concerned with the modeling of system processes and information in a technologically neutral manner. The Shared Information Model (or SID) [9], an outcome of the System view, defines the shared data and information elements of NGOSS, addressing the information and communication service industry's need for shared information/data definitions and models. The Implementation view focuses on how to build hardware, software, and firmware that will implement the system being designed. This view uses a particular NGOSS architectural style to map from the technologically neutral specification of the system to the selected target architecture.
Figure 1 (b) shows the detailed architecture of NGOSS TNA in the System view of NGOSS framework in Figure 1 (a). The NGOSS TNA [5] is the basic concept and component of NGOSS architecture. The core components of TNA are common communications vehicle (CCV), services, and contract. The service modules can communicate with each other through CCV [5], which is a generalized message bus that is independent of a specific technology. The CCV is responsible for the transport of information among application objects. The services are mainly divided into two parts: business services and framework services [5]. Business services provide the application level functionality that directly supports the implementation of a business process such as SLA management, billing mediation, QoS, etc. Framework services provide the infrastructure necessary to support the distributed nature of the NGOSS TNA. For example, they include naming and directory services, messaging services, and transaction management/monitoring

services.
The mandatory framework services consist of repository, registration, location and naming services. The registration service provides services and functionality required to support location transparency. The repository service provides a logical view of all information on a deployed distributed system. The naming service is responsible for generating and resolving unique names for the various entities contained in the repository. The location service, often built on the naming service, provides a means to map a request for an object to its particular instance.
An NGOSS contract is the fundamental unit of interoperability in an NGOSS system. A contract is used to define a specification of a service to be delivered, as well as to specify information and code that implement the service. In short, a contract is a way of reifying the specification, and implementing the functionality of the service including obligations to other entities in the managed environment. Thus, it is much more than a container of data or a specification of a set of methods.
The insulation of system architecture from technology specific details provides a number of related benefits [5]. First, it ensures the validity of the NGOSS architecture over time by supporting the deployment of new technologies without having to re-architect the entire OSS solution. Second, it provides the architectural underpinnings for the simultaneous use of multiple technologies in a single integrated OSS environment, supporting legacy systems and enabling technology migration over time. Finally, insisting that the architecture remains technology-neutral helps to prevent system design decisions from being taken too early in the development lifecycle. An excess of design detail in the core of the architecture would make it more difficult to find technologies for implementation.
The core architectural principles to be examined for the alignment of the NGOSS TNA and TSA are as follows. First, the architecture needs to provide distribution support. The architecture provides communication between business services and between process control and other system services. Second, the architecture should support the separation of business process from software implementation. Finally, the OSS needs to provide shared information for management information model. To design a technology-specific architecture, the technology must align with these NGOSS architectural principles.

II.2. Technology Specific Architecture

TMF has applied two technologies, namely XML [6] and CORBA [7], to the NGOSS TNA and specified technology application notes. The OSS/J initiative [8] proposed Java technology as a specific technology for TNA. The application notes present requirements of NGOSS TNA, overviews of each technology, and a guideline to direct technology maps to the concepts of the TNA. In this section, we present the alignment of NGOSS TNA and compare three TSAs using specific technologies: XML, CORBA, and Java. Table I [3] shows the comparison result of three technologies in accordance with the requirements of NGOSS TNA.
XML has a natural affinity with communication management. Using XML to validate, manipulate, and define the structure of application specific information models using general-purpose tools is an attractive possibility. However, XML has a substantial overhead associated with the text-only encoding of data. Also, XML/HTTP solutions suffer from the lack of availability of common distributed processing support services provided by more mature platforms such as CORBA and J2EE services [6]. CORBA is a distributed processing platform and thus supports the communication method and framework services for distributed processing. It also supports the interface definition with specifying CORBA IDLs.

However, CORBA does not provide information modeling, hence it can define the shared information using XML or other languages [7]. J2EE directly implements the principles of the NGOSS TNA such as the distribution support and separation of business process from implementation. However, J2EE architecture does not have any explicit support for the concept of shared information or federated information services as defined in NGOSS TNA [8].

Alignment	**XML**	**CORBA**	**Java**
Communication	HTTP, SOAP	GIOP, IIOP	RMI-IIOP, JMS
Framework services	➢ Runtime discovery service: UDDI ➢ Runtime location of information: XPath, XLink, XPointer	CORBA services (naming, trading, etc.)	Runtime discovery service: JNDI
Business process management	Not specified	Define new services	Define business process component
Contract interface	XML definition	CORBA IDL	Java interface
Shared information	XML Schema (information adaptation: XSLT)	Can be defined by IDL	Not specified

Table I. Comparison of Specific Technologies: XML, CORBA and JAVA

III. Web Services-based NGOSS TSA

In this section, we examine the principles of NGOSS TNA and alignments with NGOSS TNA using Web Services, and propose our Web Services-based TSA. We focus on application methods of Web Services technologies to the principles and components of NGOSS TNA.

III.1. Alignments with NGOSS TNA using Web Services Technologies

In this section, we examine the Web Services technologies such as WSDL, SOAP, UDDI, and WS-BPEL, and how they are mapped, satisfying the principles of NGOSS TNA mentioned in Section II.1.

Web Services technologies are best described as a framework of self contained, modular applications that can be discovered and executed over the network by remote programs. Web Services allow searching for services, and listing of the services and contract details for further information using the Web Services registry, normally based on the Universal Description, Discovery and Integration (UDDI) [14]. The detailed information of the services is written in Web Services Description Language (WSDL) [12]; an XML document format for describing Web Services. The requests and responses between client and remote Web Services are generally carried out by the exchange of SOAP [13] messages protocol. Web Services business process execution language (WS-BPEL) [15] is used to describe service flows which show how service-to-service communications, collaborations, and flows are performed. These Web Services technologies are applied to NGOSS TNA components such as CCV, framework services and contract to meet the principles of NGOSS TNA.

The information model is designed to be more than just a standard representation of data – it also defines semantics and behavior of, and interaction between, managed entities. The NGOSS SID model [9] provides the industry with a common vocabulary and set of information/data definitions and relationships used in the definition of NGOSS architectures. The XML Schema allows us to define the structure of management information with richer definitions of data types of XML documents including complex and data centric types. The XML Schema also

allows inheritance relationships between elements. Based on our defined XML Schema, we can define management operations and messages written in WSDL. By representing OSS/BSS data as XML, the myriad data formats used in any given OSS/BSS implementation can be integrated without requiring changes to the current system data models. With data integrity errors at the root of many OSS/BSS problems, one benefit of using XML is the framework it provides for executing sophisticated data validation and conversion from one model to another.
The second requirement of the NGOSS TNA is to provide distribution transparency. The NGOSS system needs a communication mechanism and repository that records the information used during system execution. This requirement is achieved by CCV and framework services consisting of registration, repository, naming, and location services. UDDI provides a comprehensive mechanism for locating services at run time by storing service interfaces in their WSDL format.
UDDI registration information is comprised of five data structure types: *butinessEntity*, *businessService*, *bindingTemplate*, *tModel*, and *publisherAssertion*. The *businessEntity* describes the business or other entity for which information is being registered. We can define service provider and consumer in management processes as the *businessEntity*. The *businessService* is the name and description of the services being published, which can be service information with service name, service publisher, etc. The *bindingTemplate* is information of the service including an entry-point address for accessing the service, which is used as operations binding information for the service access. The *tMotel* is a collection of information that uniquely identifies the service specification. We can use the *tModel* to store contract information, provide top-level searches of contract based *tModel* data structure and define a contract in a WSDL format. We do not use the *publisherAssertion*, which is a relationship structure that associates two or more *businessEntity* structures.
Service providers can use UDDI to register and advertise the services they offer. Service consumers can use UDDI to discover services that suit their requirements and obtain the service metadata needed to consume those services. Application programs can search the UDDI repository to find the interface that they require, download the WSDL description and use the binding information to communicate with the interface over a suitable communication channel. UDDI supports basic functionalities of framework services such as repository, registration, and location services. Thus, UDDI can be served as framework services of NGOSS TNA.
The requirement of separating business process from software implementation is achieved by the definition of contract and process. The contract, which is a definition of the service specification and message specification between service components, can be defined in WSDL based on the management information model in XML Schema format. WSDL is an XML document format for describing Web Services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented messages. The operations and messages are described abstractly, and then bounded to a concrete network protocol and message format to define an endpoint. We can define functional parts of the contract including input, output, conditions, etc. with WSDL. WSDL is sufficiently extensible to allow description of endpoints and their messages regardless of the message formats or network protocols used for communication.
The business process can be defined by WS-BPEL with the concept of process entities, process sequences, and interaction messages. Based on XML, WS-BPEL is now emerging and widely used in the industry as the standard for defining and executing business processes. Thus, we choose WS-BPEL as the process sequence definition. WS-BPEL includes four major sections in defining a business process: The <*partnerLink*> section defines the different parties that interact with

the management process in the course of processing the configuration. The <*variables*> section defines the data variables used by the process. Variables allow processes to maintain state data and process history based on the exchanged message. The <*faultHandler*> section defines the activities that must be performed in response to faults resulting from the invocation of services. The <*sequence*> section allows defining a collection of activities to be performed sequentially in a lexical order.
We can define the relations among the systems, management processes, management messages, and exception handlings using WS-BPEL. The service provider and consumer can be defined in the <*partnerLink*> section, and the contract with management operations, messages and conditions in a WSDL format can be defined in the <*variables*> section. The management sequences in accordance with the management scenario are defined in the <*sequence*> section. Due to BPEL-enabled tools, designing, testing and deploying business processes have become much simpler, even for nonprogrammers. This means business processes can be easily modified, making the operator more agile at automating costly operations or leveraging new business opportunities.
In general, Web Services are XML-based technologies that connect systems via Web. Thus, Web Services-based TSA is technically not much different from the application note of XML, which does not describe the details of applying XML technologies to NGOSS TNA.

III.2. Architecture

Web Services include many standard specifications. In Section III.1, we examined the alignments with NGOSS TNA using Web Services technologies. We focus more on the NGOSS specification from a Web Services perspective. Figure 2 shows our Web Services-based TSA. Our architecture is extended from the TNA architecture using Web Services technologies. SOAP is used as CCV to communicate between process entities, and WSDL is used to define contracts between process entities through SOAP. UDDI supports the framework services of repository, registration, naming, and location services while WS-BPEL supports the process service that executes the management functions including framework services.

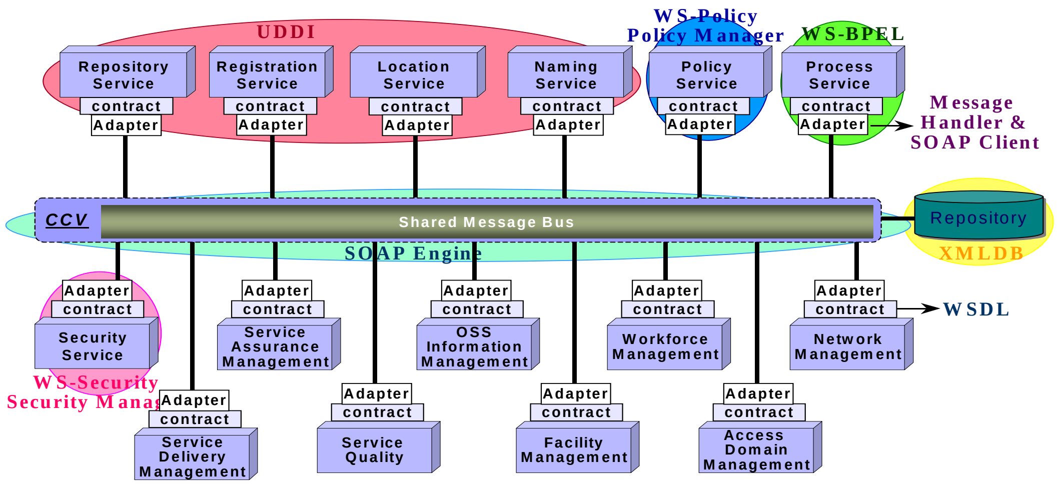


Figure 2. Web Services-based TSA

The policy service can use the definition of WS-Policy [16] as the definition of information and policy manager as the engine. The security service can use the WS-Security specification [17] as the definition of information and security manager as the service engine. The management operation services such as service assurance, service delivery and so on, can be defined as new services using WSDL. The adapter acts as a message handler with a SOAP client module while XMLDB can be used as the management repository instead of the traditional RDBMS. The native XMLDB stores the data structured as XML without having to translate the data to a relational or object database structure, which is especially valuable for complex and hierarchical XML structures that would be difficult or impossible to map to a more structured database. However, the XMLDB is not mandatory due to the performance issues of processing XML document.

IV. Practical Use of Web Services-based TSA

In this section, we present a case study to verify our technology specific architecture. We have selected the process of DSL fulfillment, which is one of the examples of eTOM's fulfillment process, as a sample case to verify our technology specific architecture. The eTOM fulfillment [10] is a vertical end-end process grouping responsible for providing customers with their requested products in a timely and correct manner. It translates the customer's business or personal need into a solution, which can be delivered using the specific products or services in the enterprise. This process informs the customers of the status of their purchase order, ensures completion on time, as well as ensuring a delighted customer. In this section, we define management information and process sequences for DSL fulfillment.

IV.1. Management Information

As explained in Section III.1, we define our management information through XML Schema based on SID modeling [9]. The SID specification defines the business entities and their characteristics including data type, description and whether they are required or optional. Figure 3 shows the XML Schema defining the information model of customer, product and customer order. The information of customer order references one customer, and one or more products. The product includes the product name, description, product number, manufacturer, valid period and lifecycle status. The customer contains the customer ID, and customer status as mandatory elements. The customer order contains all contents of the customer and product and refers to each type defined as the complex type of customerType and productType, respectively. The customer order also contains an assigned priority (order handling priority) and assigned response date (order response date) as attributes. We define all management information such as service parameter, service order, resource parameter, and resource order followed by the SID modeling [9].

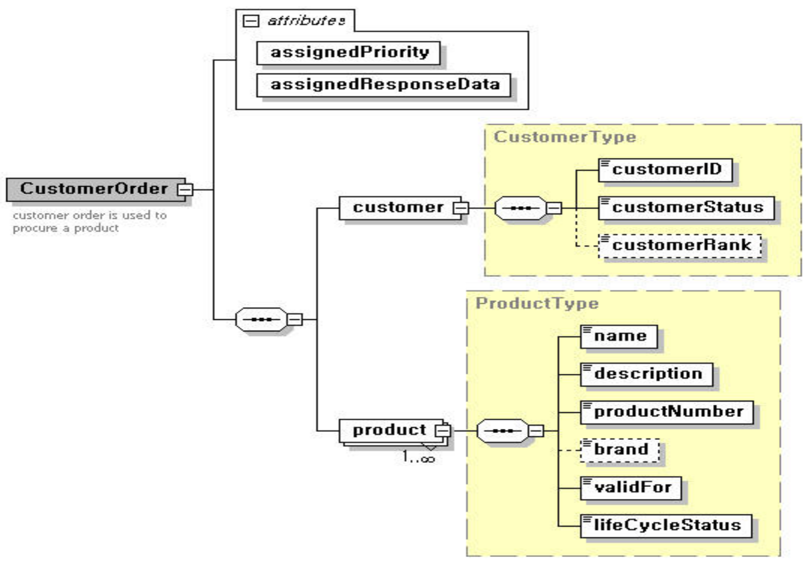


Figure 3. Shared Information of DSL Fulfillment

IV.2. Management Process

There are *Order handling* module, *Service configuration & activation* module, and *Resource provisioning* module in the management system for DSL fulfillment. The *Order handling* module receives the customer order request of DSL service from a customer and then sends the service order request to the *Service configuration & activation* module. The *Service configuration & activation* module requests the resource provisioning to the *Resource provisioning* module in order to acquire the resource after the service configuration in accordance with the service request. After acquiring the resource, the *Service configuration & activation* module activates the service and reports the result of service activation to the *Order handling* module.

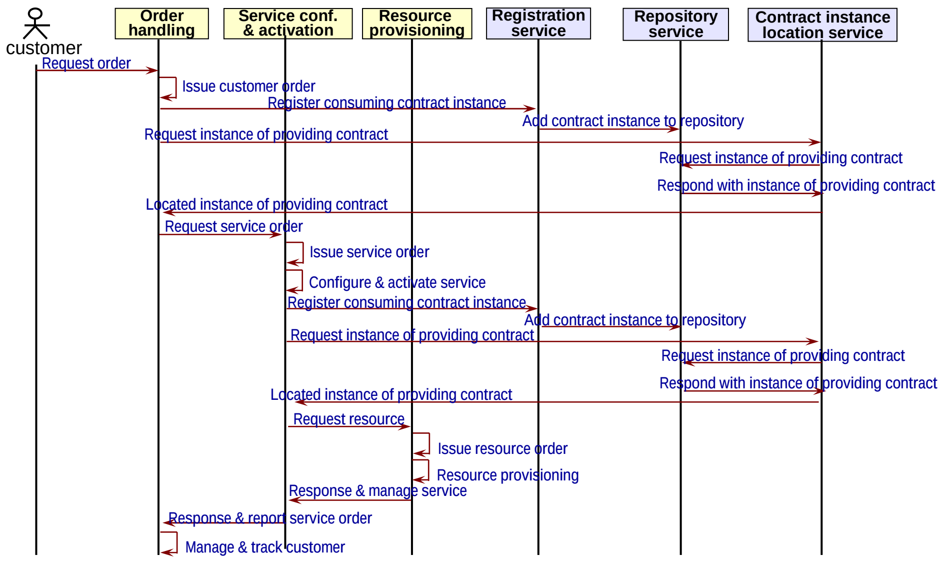

customer
Order handling
Service conf. & activation
Resource provisioning
Registration service
Repository service
Contract instance location service
Request order
Issue customer order
Register consuming contract instance
Add contract instance to repository
Request instance of providing contract
Request instance of providing contract
Respond with instance of providing contract
Located instance of providing contract
Request service order
Issue service order
Configure & activate service
Register consuming contract instance
Add contract instance to repository
Request instance of providing contract
Request instance of providing contract
Respond with instance of providing contract
Located instance of providing contract
Request resource
Issue resource order
Resource provisioning
Response & manage service
Response & report service order
Manage & track customer

Figure 4. Management Scenario of Service Order

Figure 4 shows the combination of interaction sequences between framework service components and the sequence for the service order. That is, there should be a process to make a contract between two modules for the *Order handling* module in order to request the service order to the *Service configuration & activation* module. In the process of making this contract, the *Order handling* module becomes a consuming contract instance, and the *Service configuration & activation* module becomes a providing contract instance. When the contract between the *Order handling* module and the *Service configuration & activation* is agreed with, the service requester (the *Order handling* module) can send the service order request to the service provider (the *Service configuration & activation* module).

```
<process name="serviceOrder">
    <partnerLinks>
        <partnerLink name="customerManager" partnerLinkType="customerManagerLT" />
        <partnerLink name="serviceManager" partnerLinkType="serviceManagerLT" />
        <partnerLink name="resourceManager" partnerLinkType="resourceManagerLT" />
    </partnerLinks>
    <variables>
        <variable name="CustomerOrderRequest" messageType="CustomerOrderRequestType" />
        <variable name="ServiceRequest" messageType="ServiceRequestType" />
        <variable name="ResourceRequest" messageType="ResourceRequstType" />
        <variable name="CustomerOrderResponse" messageType="CustomerOrderResponseType" />
        <variable name="ServiceResponse" messageType="ServiceResponseType" />
        <variable name="Faullt" messageType="FaultType" />
    </variables>
    <faultHandler>
        <catch faultName="cannotFindCustomer" faultVariable="Fault" faultMessageType="FaultType">
            <reply partnerLink="customerManager" portType="requestCustomerOrderPT"
         operation="sendRequestCustomerOrder"
              variable="Fault" faultName="cannotFindCustomer" />
        </catch>
    </faultHandler>
    <sequence>
        <receive partnerLink="customerManager" portType="requestCustomerOrderPT"
         operation="sendRequestCustomerOrder" variable="CustomerOrderRequest" />
        <sequence>
             <invoke partnerLink="customerManager" portType="requestCustomerOrderPT"
              operation="requestCustomerOrder" inputVariable="CustomerOrderRequest"
              outputVariable="CustomerOrderResponse">
        </sequence>
        <sequence>
             <assign> <copy> <from variable="requestCustomerOrder" part="serviceOrder" />
                                   <to variable="ServiceRequest" part="serviceOrder" /> </assign>
            <invoke partnerLink="serviceManager" portType="requestServicePT" operation="requestService"
                         inputVariable="ServiceRequest" outputVariable="ServiceResponse"> </invoke>
        </sequence>
        <sequence>
           <assign> <copy> <from variable="ServiceRequest" part="serviceOrder" />
                                   <to variable="ResourceRequest" part="serviceOrder" /> </assign>
           <invoke partnerLink="resourceManager" portType="reuqestResourcePT" operation="reuqestResource" />
         </invoke>
       </sequence>
        <reply partnerLink="customerManager" portType="requestCustomerOrderPT"
         operation="sendRequestCustomerOrder" variable="CustomerOrderResponse"> </reply>
    </sequence>
</process>
```

(a) Management Scenario: WS-BPEL

```
<Contract>
  <General> ... </General>
  <View name='System_View'>
    <Functional>
      <Associated_System_Processes> manage_track_customer, issue_customer_order </Associated_System_Processes>
      <Associated_System_Policies> </Associated_System_Policies>
      <System_Capabilities>
        <Input_Entities> CustomerOrderRequest </Input_Entities>
        <Output_Entities> CustomerOrderResponse </Output_Entities>
        <Pre-Conditions> customer_is_registered, request_is_feasible_to_networkResources
        </Pre-Conditions>
        <Termination> Successful </Termination>
        <Post-Conditions> CustomerOrder_is_issued_accordingly </Post-Conditions>
        <Post-Conditions_System_Exceptions> None </Post-Conditions_System_Exceptions>
        <Interaction_Points> manage_track_customer, issue_customer_order </Interaction_Points>
        <Interaction_Roles> CustomerManager</Interaction_Roles>
        <Security> None </Security>
        <Context> This contract is used to progress handling of customer order and issue the order </Context>
      </System_Capabilities>
    </Functional>
    <Non-Functional> ... </Non-Functional>
    <Management> ... </Management>
  </View>
<Contract>
```

(b) Contract Example: RequestCustomerOrder

Figure 5. Service Order: WS-BPEL Definition and Contract

Figure 5 (a) defines the sample scenario in Figure 4 using WS-BPEL. First, we define the process entities in the *partnerLinks*, messages from senders and receivers in the *variables*, and management faults in the *faultHandler*. We also define the process sequence that is executed when the messages from some *partnerLinks* arrive in the *Sequence*.
We have defined the contract of the system view in the XML format. The most important part in five parts of contract (General, Functional, Non-functional, Management and View specific) is the functional part, in which input, output variables and conditions need to be defined. Figure 5 (b) describes the system view contract of 'RequestCustomerOrder' between the *Order handling* module and the *Service configuration & activation* module for the DSL fulfillment in the XML

format. The *General* part contains Contract names, Versions and explanations while the *Functional* part contains Pre-condition, Post-condition, Input & Output Parameters and so on. The *Non-Functional* and *Management* parts need to be defined when it is necessary to extend the capability of QoS, policy, cost and management.

V. Implementation of Web Services-based TSA

In this section, we present the implementation details of our Web Services-based TSA. We have implemented TNA components and management functions using Web Services technologies based on our proposed design and named our OSS system as Web Services-based OSS (WS-OSS).
Our WS-OSS system has been implemented on a Linux server. We used the Apache Tomcat 4.0 for the Web server and Servlet engine. The Apache Web Services project package [16], which is Java-based Web Services software, is also used. We used jUDDI 0.9rc4 for a UDDI implementation, Axis 2 v1.0 for a SOAP engine, Apache Addressing for a WS-Addr handler, and Pubscribe 1.0 for a Web Services notification handler. In addition, Agila BPM is used for a BPEL engine and Xindice v1.0 for an XML DB. Figure 6 shows the implementation architecture of our WS-OSS system.

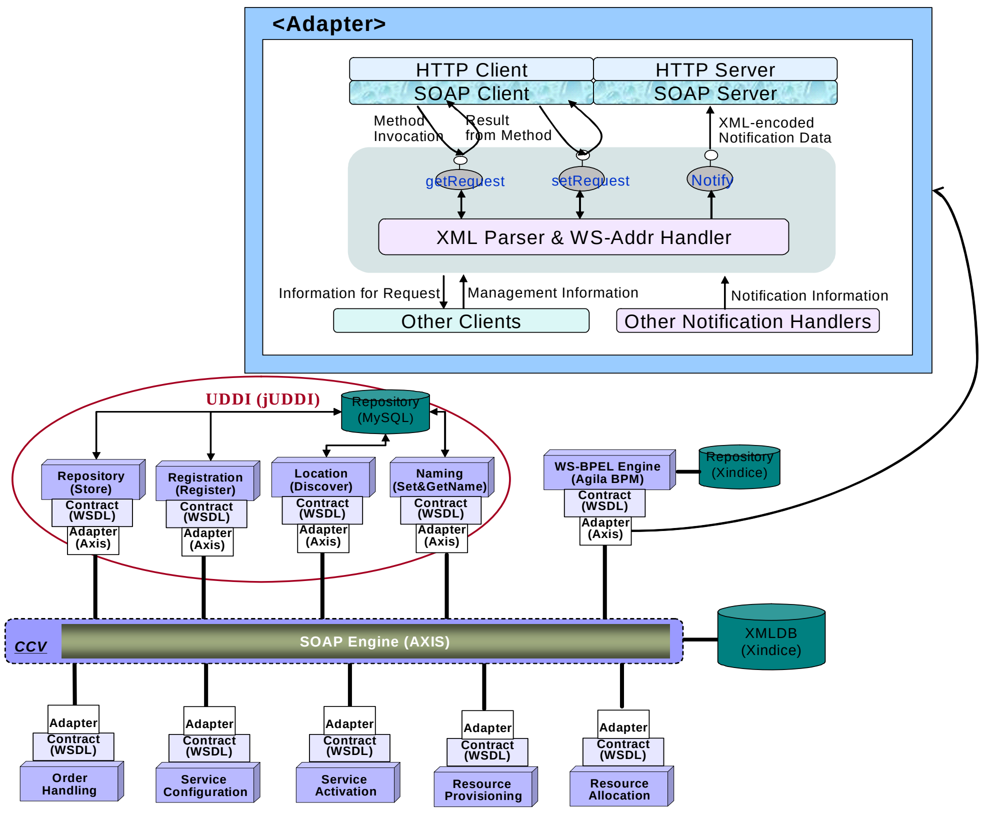


Figure 6. Implementation Architecture

The *Adapter* module of the rectangle in Figure 6 acts as a gateway for communication and message handling. The *Adapter* is implemented with Axis package; it uses a SOAP client module for requesting data to the other service modules and a SOAP server module for receiving notification from other service modules. In our implementation, the roles of CCV are distributed into the *Adapter* module. Using this adapter approach, we can integrate the existing OSS system into our WS-OSS with other types of clients and notification handlers. In the case

of the same management protocol (SOAP request), direct requests without going through other types of clients and notification handler are performed only through the SOAP client and server. In our WS-OSS system, we concentrate more on framework services and TNA components such as CCV and contract. We do not consider the policy and security services modules.
We first implemented the framework services using existing UDDI APIs. The naming service checks the registration information and service name using the *get_registeredInfo* API, which provides a summary of all information registered on behalf of a specific user. The repository service provides an update of service information using publisher APIs such as *save_business*, *save_service*, *save_binding*, *save_tMode*l, *delete_business*, *delete_binding*, etc. The registration service calls the repository service to register for the provided and consumed services. If it is necessary to check the naming, then it calls the naming service. The location service provides service information search using inquiry APIs such as *find_business*, *find_service*, *find_binding*, *find_tModel*, *get_businessDetail*, *get_serviceDetail*, etc. We omitted the request from the location service to repository service to inquire service information in Figure 4 by directly providing the search capability to the location service. This can reduce the operation calls and loads of the repository service.
Then, we deployed these framework service functions into Web Services using a Java Web Service (JWS) provided by Axis. This deployment method automatically generates a WSDL file and proxy/skeleton codes for the SOAP RPCs. In order to enable the existing Java class as a Web Service, we simply copied the Java file into the Axis Web application, using the extension '.jws' instead of '.java'. We implemented framework services such as repository, registration, location and naming services using this mechanism. Unlike this mechanism, we first defined WSDLs, generated Java stubs, and filled the codes in the stubs. We implemented other services, namely, management function modules such as order handling, service configuration, etc., using the second method. Finally, we defined the business process sequence in WS-BPEL, in which the defined process flows were stored in their own process repository, and Agila BPM engine executed the process in accordance with the WS-BPEL definition.
The execution step of our order handling for a DSL fulfillment is as follows: A customer orders new DSL service through the service application form of Web-based user interface. This request message is handed over to the BPEL engine, and the engine checks the request message and the partnership. Then, it directly calls the order handling service. The management processes are conducted in the sequence of Figure 4 and the BPEL engine processes these management sequences. The contracts between framework services are interface-level, which are operations in a WSDL format called by each other. The contracts between the management processes are registered and stored via registration and repository services, respectively, and preserved in the repository of UDDI itself.

VI. Performance Evaluation

In this section, we present performance evaluation results of our system. We evaluate the processing time of each TSA component and determine the bottleneck component. We also propose a method to reduce the processing overhead of the bottleneck component.

VI.1. Performance Test Environment

Figure 7 shows our performance evaluation environment. We independently implemented a jUDDI server for the services registration, an Xindice DB server for information repository, and a server for service consumers, service providers, and a BPEL engine. We processed the DSL fulfillment in Figure 4. As shown in the sequence diagram of Figure 4, an actor can be a service provider or a service consumer, thus we implemented the service providers and consumers in the server. As mentioned in Section V, the servers were implemented on Linux OS with a Pentium-III 2.4 GHz CPU and 1GB RAM using Java language and connected via 100Mbps LAN.

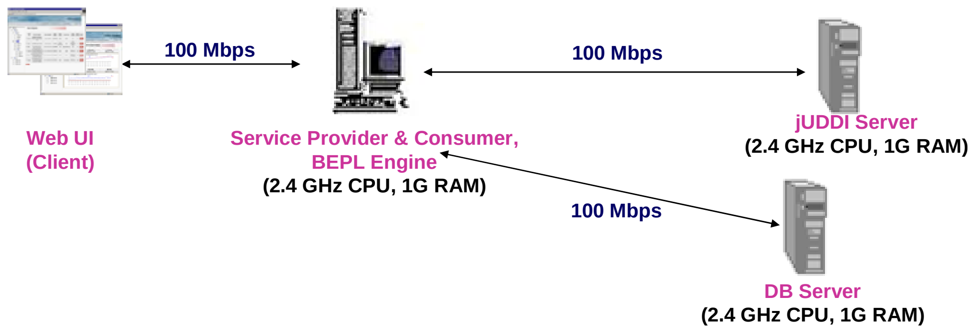

100 Mbps
100 Mbps
Web UI (Client)
Service Provider & Consumer, BEPL Engine (2.4 GHz CPU, 1G RAM)
jUDDI Server (2.4 GHz CPU, 1G RAM)
100 Mbps
DB Server (2.4 GHz CPU, 1G RAM)

Figure 7. Performance Evaluation Environment

VI.2. Performance Test Result

In this section, we examine an NGOSS TSA component which generates the biggest processing overhead. The Web Services-based NGOSS TSA is mainly composed of components such as a BPEL engine, a UDDI engine, and a DB server. We increase the concurrent service requests from 1 to 100 by 25 and investigate the processing time of each component. The concurrent requests are generated using a thread mechanism.

Figure 8 shows the processing overhead of each component in accordance with the number of requests. As the number of requests increments, the UDDI engine which processes framework services becomes the bottleneck component. As mentioned in Section III.1, each part of contract is mapped to the UDDI data structure and four UDDI data structures such as butinessEntity, businessService, bindingTemplate, and tModel need to be registered to the UDDI registry to define only one contract. Moreover, the search of a contract also needs to investigate four UDDI structures. Thus, the UDDI engine generates more processing overhead as the number of simultaneous requests increases. In the case of only one customer's request, the processing time of BPEL engine occupies the biggest of the entire processing time. However, as the number of requests increases, the portion of processing time of BPEL engine in the entire processing time relatively reduces compared to the processing time of UDDI engine. The processing time of XMLDB is comparatively low because most DB operations to process DSL fulfillment are insert operations to add customer's order and correspondent service information. The insert operation is relatively shorter than the query operation in XMLDB.

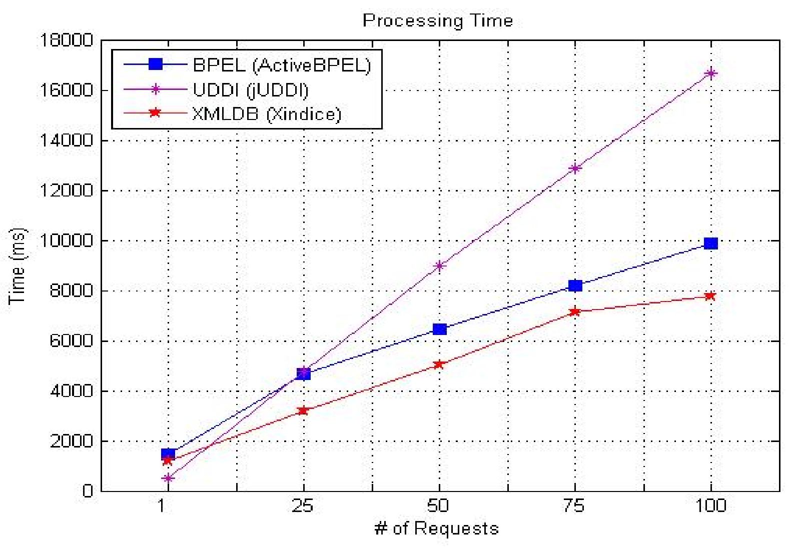


Figure 8. Processing Time per Component

VI.3. Performance Enhancement Method

In this section, we investigate the performance enhancement method of processing time of UDDI registration and search, which generates the biggest processing overhead. To reduce the processing overhead of UDDI search, we use a cache mechanism. Instead of sending UDDI search operations to the jUDDI engine every time the service consumer requests, we can search the information for the same service in the cache. This reduces the search overhead of UDDI registry. Figure 9 shows the comparison result of processing time of UDDI search with and without the cache mechanism. As the result of applying the cache mechanism to the UDDI search, we gain 25% performance enhancement of processing time of UDDI. The enhancement by using the cache mechanism is obtained only in the search part, not the registration part. To reduce the processing overhead of registering contracts, we also need to devise a method to merge many contracts into one.

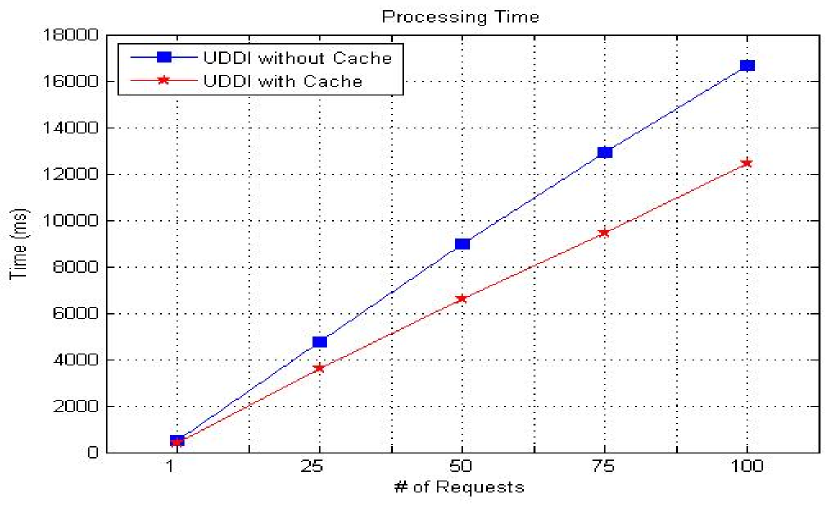


Figure 9. Performance Enhancement with Cache Mechanism

To decrease the overhead of XMLDB, we consider the use of a relational DB such as MySQL. We compared the processing time of Xindice DB and MySQL DB in terms of insert and query operations. Figure 10 shows the processing time of adding and querying customer information to DB. We performed insert operations as the number of concurrent requests increased from 1 to 200 by 50. The processing time of insert operation in MySQL DB is roughly twice as faster than that in Xindice DB. The processing time of query operation in MySQL slightly increase as the number of insert request advances and remains almost static regardless of the number of existing data. However, the processing time of query operation in Xindice increases largely as the number of data in DB advances. This is because Xindice uploads the entire XML data into memory to parse. Thus, Xindice is not efficient when we need to process query operations with lots of data. Note that, this performance result is merely limited to Xindice XMLDB of the Apache group.

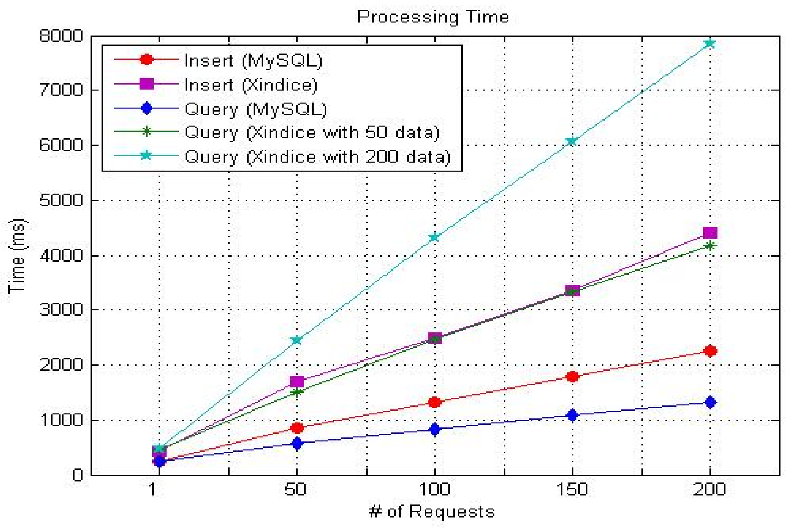


Figure 10. Performance Comparison of Xindice and MySQL

VI.4. Experience and Discussion

Web Service satisfies the characteristics of NGOSS SOA and supports COTS and freely available tools that can be easily applied to NGOSS TNA components. Specifically, the management information using XML Schema supports powerful information definition. The contracts including service processors, pre/post conditions and input/output messages are also defined in WSDL. UDDI is used as framework services and SOAP acts as CCV. Moreover, the management scenario can be defined in WS-BPEL and processed in the BPEL engine. Partners in BPEL definition are acted as service providers and consumers in UDDI repository; operations and messages in BPEL are mapped to messages and operations in WSDL. The messages in WSDL are also mapped to actions in SOAP. In this manner, Web services technologies applied to each NGOSS TSA component are connected and operated together. Consequently, Web Services technologies fully support the principles and functionalities of NGOSS TNA. That is, it is important to understand the relationship between the elements of each technology and define the elements to organically operate services.
We have used four data structures of UDDI such as butinessEntity, businessService, etc., to define the contract and implemented framework services using UDDI APIs such as get_registeredInfo, save_business, save_servic, find_business, find_servic, etc. To register a contract, we need to register four data structures by calling four save operations per each data structure, which generates much processing overhead of service registration. This overhead is the same as the search operations. Thus, we have applied a cache mechanism to search operations of UDDI registry to reduce the search processing overhead. But, this is only a temporary solution. Ultimately, we need to revise the mapping of

contract to UDDI data structure towards defining a contract using one or two UDDI data structures. Also, we can use a relational DB instead of XML DB to decrease the overall processing time. XMLDB is helpful to manipulate to data represented in XML format, but, requires much processing time and computing power to parse XML data.

VII. Concluding Remarks

In this paper, we examined the concept, architectural principles and components of the NGOSS TNA. We proposed an NGOSS technology-specific architecture using Web Services technologies in accordance with the principles of NGOSS TNA. Web Service is a distributed and services-oriented computing technology that can be applied to the NGOSS TNA. We examined the advantage of applying Web Services technologies to NGOSS TNA. To validate our proposed architecture, we implemented a prototype of Web Services-based TSA focusing on TNA components such as CCV, framework services, and contract and management functions using the DSL Service Order Fulfillment example. We also presented performance evaluation results, determined a bottleneck component in our system and proposed a performance enhancement method using the cache mechanism. Our work can be used as a guideline on how Web Services technologies are applied to the NGOSS components, for anyone planning to develop a Web Services-based NGOSS TSA.
We are currently extracting the performance metrics of Web Services-based TSA and conducting performance analysis. We need to implement other service modules such as policy and security services and will also extend our system with more management functions based on eTOM operations.

Acknowledgments

This research was supported in part by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Cen ter) support program supervised by the IITA (Institute of Information Technolo gy Assessment) (IITA-2006-C1090-0603-0045), and by the Electrical and Com puter Engineering Division at POSTECH under the BK21 program of the Minist ry of Education, Korea.

References

[1] TM Forum, TM Forum, http://www.tmforum.org/, Refer Oct. 2007.
[2] Georgalas (N), Azmoodeh (M), Using MDA in Technology-independent Specifications of NGOSS Architectures, *1st European Workshop on MDA (MDA-IA 2004)*, Enschede, The Netherlands, pp. 11~18, Mar. 2004.
[3] Choi (M), Ju (H), Hong (J), Yun (D), Design of NGOSS TSA using Web Services Technologies, *Proc. of Integrated Network Management Symposium (IM2007)*, Munich, Germany, pp. 868~71, May 2007.
[4] TM Forum Technical Program, New Generation Operations Systems and Software (NGOSS), RN303 R6.0, Jun. 2006.
[5] TM Forum, NGOSS Technology Neutral Architecture, TMF053 v5.3 r6.0, Nov. 2005.
[6] TM Forum, NGOSS Phase 1 Technology Application Note-XML, TMF057 v1.5, Dec. 2001.

[7] TM Forum, NGOSS Phase 1 Technology Application Note - CORBA, TMF055 v1.5, Aug. 2001.
[8] Ashford (C), OSS through Java as an Implementation of NGOSS, White paper, Apr. 2004.
[9] TM Forum, Shared Information/Data (SID) Model, GB 922, Release 6.0, Nov. 2005.
[10] TM Forum, Enhanced Telecom Operations Map (eTOM), GB921, Release 6.0, Nov. 2005.
[11] W3C, Web Services Architecture, W3C Architecture WG Notes, Feb. 2004.
[12] W3C, Web Services Description Language (WSDL) 1.1, W3C WSDL WG Note, Mar. 2001.
[13] W3C, SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation, Jun. 2003.
[14] OASIS, UDDI version 3.0.2, UDDI Spec Technical Committee Draft, Oct. 2004.
[15] Business Process Execution Language for Web Services Version 1.1. Second Public Draft Release, BEA Systems, International Business Machines Corp., Microsoft Corp., SAP AG, Siebel Systems, May 2003.
[16] W3C, Web Services Policy 1.5 - Framework, W3C Candidate Recommendation, Oct. 2007.
[17] OASIS, Web Services Security: SOAP Message Security 1.1 (WS-Security 2004), OASIS Standard Specification, Oct. 2006.
[18] Apache Software Foundation, Web Services Project @ Apache, http://ws.apache.org, Refer Oct. 2007.

Figure List

Table List