MODELING AND SIMULATION AS A SERVICE

A Thesis Presented to the Department of
Computer Science,
African University of Science and Technology, Abuja

In Partial Fulfillment of the Requirements
For The Degree of

MASTER OF SCIENCE

By
MainroalNgargoto Rolland

Abuja, Nigeria.

December, 2014.

MODELING AND SIMULATION AS A SERVICE

By

MainroalNgargoto Rolland

A THESIS APPROVED BY THE COMPUTER SCIENCE DEPARTMENT

RECOMMENDED: ……..

Supervisor, Professor Mamadou Kaba Traoré

……..

……..

Head, Department of Computer Science

APPROVED: ……..

Chief Academic Officer

..……

Date

ABSTRACT

DEVS Modeling & Simulation separates a model from its simulator. A DEVS model describes the structure and the behavior of a system, while a DEVS simulator generates the trajectories of these descriptions through execution threads.

The goal of a DEVS standard is to provide a simple and mostly automated way of executing simulations that involve remote and/or heterogeneous DEVS models. This can be achieved by taking two different approaches, which are simulator-based interoperability and model-based interoperability:

- In model-based interoperability, models themselves are deployed as services instead of simulators. Using this model driven approach, the operations invoked through the network are no longer simulation mechanisms, but model functions.
- In simulator-based interoperability, the main idea is to have a collection of simulation services distributed over the Internet. These services provide several operations for simulating DEVS models in a unified manner.

In this work,we propose a Web Services-based modeling and simulation which can be used to solve the above mentioned problem of interoperability of DEVS-based models and DEVS-based simulator. We thendeployed an example of Web Services-based modeling and simulation using the DEVS toolkit SimStudio_1.1and Apache Axis2. We also did a literaturereview of SOA-based DEVS modeling and simulation.

Acknowledgement

There are many people who have helped me directly or indirectly with this thesis work. I am particularly grateful to my supervisor, Prof. Mamadou Kaba TRAORE for introducing me to this subject and many instructions on this subject. I appreciate his guidance, encouragement, and support during the period I worked on this thesis.

I would also like to thank all AUST Staff (Academic and non-Academic) for their contribution during our M.Sc. Courses.

My colleagues at African University of Science and Technology have been very wonderful. Special thanks to DAVID ADELANI for several discussions on my thesis work and for his excellent suggestions. Thanks also to TIAKO FANI, OGBONNA, SALAMI, POPOOLA, GONDWE, MANTHALU, AGBONKINA, ADEKUNLE, ADEBAJO. Thanks also to PhD Students at AUST, especially Computer Science PhD.

I would also like to thank my dear fiancée, CLAUDINE VADMI for being present in my life though we are far apart with numerous encouragement and support.

My great family has been very supportive. Thanks to my elder brother and sister, TOUDJINGAR GOUMANTAR FELICIEN and DENEYAMBAYE ALBERTINE, and my mumfor their financial assistance, prayers and encouragement.

My greatest thanks go to the Almighty God, to whom I owe my life, for His benevolence, mercy, and love.

This thesis work was funded by the African University of Science and Technology.

Dedication

To my late father DEONODJI NGARGOTO GOUMANTAR.
His words of inspiration and encouragementin pursuit of excellence, still linger on.

Table of Contents

Chapter 1: Introduction....10
I.1 Research Context....10
I.2 Problem Statement....10
I.3Research Objectives....10
I.4 Research methodology....10
I.5 Organization of the document....11
Chapter 2: Related work....12
2.1 Standardizing DEVS Model Representation....12
2.2 DEVS/SOA....12
2.3 DEVS Modeling Language (DEVSML)....13
Chapter 3: DEVS and Web services State of Art....14
3.1 DEVS....14
3.1.1 DEVS modeling formalism....15
3.1.1.1 Classic DEVS atomic model....15
3.1.1.2 Classic DEVS coupled model....16
3.1.1.3 PDEVS atomic model....17
3.1.1.4 PDEVS coupledmodel....18
3.1.2. DEVS toolkits....18
3.2Web services....19
3.2.1 Service-Oriented Architectures (SOA)....19
3.2.2 Existent technologies-based SOA....21
3.2.2.1 CORBA....21
3.2.2.2 JAVA RMI....21
3.2.2.3 DCOM....22
3.2.3Web Services....23
3.2.3.1 Web Services components....24
3.2.3.2 Service Description: WSDL....25
3.2.3.3 Service Discovery (UDDI)....25
3.2.3.4 Why Web Services....26
3.2.3.5 Web services Tools and Vendors....27
3.2.3.6 Example of services....27
Chapter 4: DEVS modeling and simulation as a service....29
4.1 Overview....29
4.2 DEVS modeling and simulation as a service....29

4.2.1 Example of DEVS modeling and simulation as a service using the SimStudio_1.1 29
4.2.1.1 Axis2 Web Services deployment of the SimStudio_1.1 30
4.2.1.1.1 Configuration of Eclipse for Web Services with Axis2 30
4.2.1.1.2 Configuration of services.xml 31
4.2.1.1.3 Creation and deployment of Service archive 34
4.2.1.2Consuming the Axis2 Web Services 35
Chapter 5 Conclusion 36
5.1 Result 36
5.2Challenges 36
5.3 Perspectives 36
References 38

Table of images

Chapter 1: Introduction

This chapter introduces the topic of this research project with its context, problem statement, objectives, and methodology. In addition it introduces the organization of the present document.

I.1 Research Context

During the last years, the DEVS community provides many contributions towards the realization of a world-wide platform for collaborative Modeling & Simulation. The goal of such a platform would be to enable the sharing and reuse of models between scientists, as well as the seamless simulation of distributed and heterogeneous models. Therefore, one of the major research fields is the definition of architectures for integrating heterogeneous DEVS components, meaning simulators and/or models written in different frameworks and programming languages. In this thesis, we present Web services, one strategy for providing such interoperability between DEVS components. Web services are concerned with the problems of enabling systematic application-to-application interactions over the Web, and the integration of the existing network computer infrastructure into the Web. The goal of Web services is to provide a flexible frameworkbased on sending XML messages in a specific SOAP format. SOAP is a specification that defines an XML grammar for both sending messages and responding to messages that you receive from other parties. The goal of SOAP is to describe a message format that is not bound to any hardware or software architecture, but one that carries a message from any platform to any other platform in an unambiguous fashion.

I.2 Problem Statement

Distribution and Interoperability among DEVS-based models and DEVS-based simulators continues to be of key interest within the simulation community[1].The reason is that DEVS Modeling and Simulation (M&S) has various implementations with various computer languages such as JAVA, C++, and C#. To enhance model reusability with different implementation, and an interoperable mechanism for simulation of heterogeneous DEVS models, we need interoperable systems such as CORBA, HLA, and SOA[2]. As an infrastructure of an interoperable system, SOA is applicable because it provides platform and language independence. In this work, we will studyhow to realize SOA using web service technology and try to see how it can be applied to enhance the problem of distribution and interoperability of DEVS-based models and DEVS-based simulators.

I.3Research Objectives

Firstly this research aims to propose a web service solution that will guaranty the interoperability between DEVS-based model and DEVS-based simulator in order to enhance the collaboration among DEVS programmers; secondlyit aims to give an example of DEVS modeling and simulation as service usingexisting DEVS toolkit SimStudio_1.1. The rest of the thesis will provide answers to the following questions:

- What technologies are used for the development of DEVS-based model and simulator?
- Why and How to create DEVS-based modeland simulator as a web service?

I.4 Research methodology

In this work, technologies such as web services and its component WSDL, SOAP, and UDDI were used to achieve the distribution and interoperability of DEVS modeling and simulation. The majority of the work was based on understanding web service. Tools such as apache-tomcat-6.0.39, eclipse-jee-helios-SR2-win32, apache axis2-1.6.2, and two (2) eclipse plugins: apache axis2_service_archiver and apache axis2_code_generator were used to illustrate the deployment of web service. We will use the DEVS toolkit SimStudio_1.1 as an example of service to offer over the internet using those technologies.

I.5 Organization of the document

This thesis is organized in the following way. Chapter 2 discusses the related work on Web services-based DEVS modeling and simulation. Chapter 3 describes the state of art of DEVS modeling and simulation, and Web services. Chapter 4 presents the sample deployment of Web service.Chapter 5 presents the conclusion.

Chapter 2: Related work

This chapter gives an overview over relevant existing work in the field of Web Service-based DEVS modeling and simulation.

2.1 Standardizing DEVS Model Representation

DEVS modeling and simulation research groups are interested in DEVS interoperability in order to enhance model composability and reusability of DEVS models in different languages and platforms.

Introduced by Wainer and al. [2, 3, 4], standardizing DEVS model representation allows a model to run on any DEVS simulation environment. As shown by Figure 1, the DEVS model created with any programming language is converted as XML file known asDEVS/XMLand saved in repository.Using XML-based eXtensible Stylesheet Language Transformations (XSLT), the DEVS/XML model is transformed as a DEVS model in a given programming language. XML have been usedas a mechanism for interchanging model information.

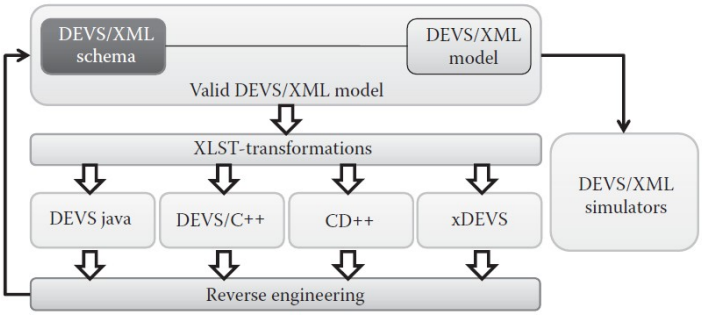


Figure DEVS/XML standard definition

Using XML for interchanging model information present a lot of advantages[6]since XML is a data interchange language of web services. In addition,XSLT offer a convenient way to specify translations of XML documents.This is powerful in the sense that a model can be retrieved from a repository and run locally on a different tool other than the originally intended running environment of that model.

The standardizing DEVS model representation is applied as a solution toenhance model composability and reusability of DEVS models in different languages and platforms. This solvespartially the problem of the distribution and interoperability of DEVS-based model and DEVS-based simulation discuss in this present work.

2.2 DEVS/SOA

DEVS Modeling and Simulation (M&S) is implemented with several computer languages such as Java, C# or C++. The need of a distributed platform to provide interoperability mechanics for

simulation and encourage reusability and integration of diversified DEVS models becomes a priority.

DEVS/SOAintroduced byS. Mittal and al. [7],following the proposed interoperability standardfor DEVS M&S is a new framework using Web Service Description Language (WSDL) and Service Object Access Protocol (SOAP) protocols defining respectively simulator and coordinator interfaces and the communication between them.

As shown by Figure 2, the major advantage using DEVS/SOAis that DEVS services implemented in different programming languages can be distributed through the Internet and connected using WSDL and SOAP standards.

The drawback in DEVS/SOA environment is that atomic model will always be downloaded to generic simulation server and locally compile. This reduces the communication with the remote servers. But it also implies the non-synchronization of the downloaded atomic model if the original one is updated.Like DEVS/XML, the DEVS/SOA solves partially the problem of the distribution and interoperability of DEVS-based model and DEVS-based simulation discuss in this present work.

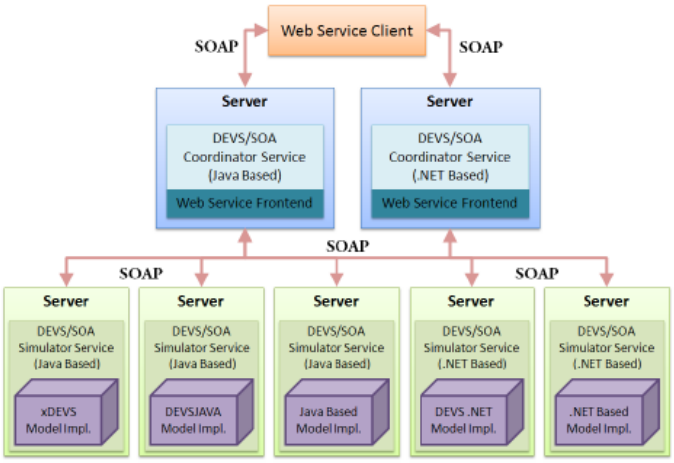


Figure DEVS/SOA Distributed Architecture

2.3 DEVS Modeling Language (DEVSML)

DEVSML proposed by B. P. Zeigler[4], is built on eXtensible Markup Language (XML) as the preferred means to provide model interoperability between the disparate simulator implementations and provides a means to make the simulator transparent to model execution. DEVSML also provides the capability to translate model to and from XML and JAVA programming language leading to model composability and validation. In other word,DEVSML enables creation of models, which can be shared between different simulation environments. Figure 3 shows the architecture of DEVSML.

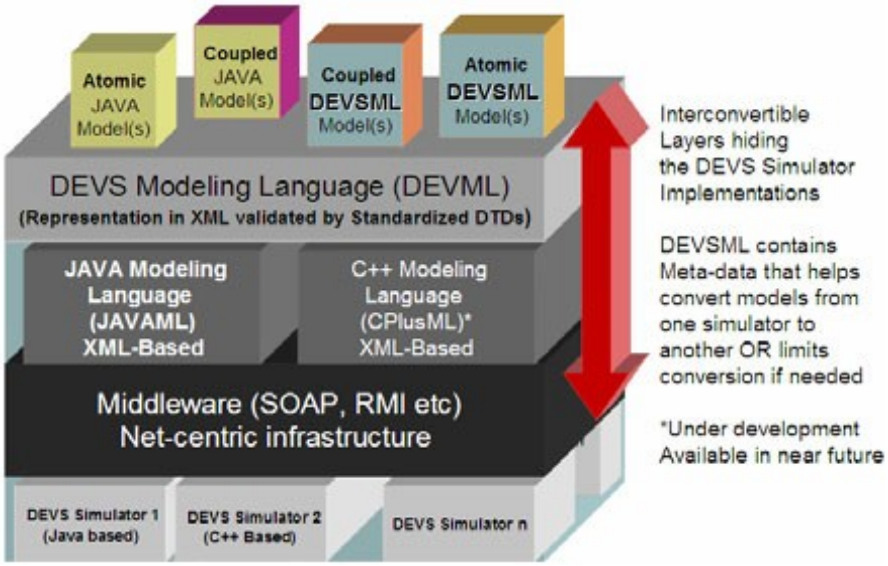


Figure DEVSML layered architecture

DEVSML partially solves the interoperability in DEVS/XML, and fully supports the reuse between different simulation platforms by means of reverse engineering. Hence it solves partially the problem of the distribution and interoperability of DEVS-based model and DEVS-based simulation discuss in this present work.

Chapter 3: DEVS and Web services State of Art

The following Chapter will detail the state of the art in the area of Discrete Event System Specification (DEVS), and Web service technologies. This Chapter is intended to explain the technologies, concepts and notions that will be used through the rest of this thesis.

3.1 DEVS

The modeling and simulation framework defines a set of entities (real system, model, simulator, and experimental frame (EF)) and their relationships (modeling relation and simulation relation) that are central to the M&S enterprise[11]. The basic entities of the framework are source system, model, simulator, and experimental frame as illustrated in Figure 4, and they are linked to the modeling and simulation relationships.

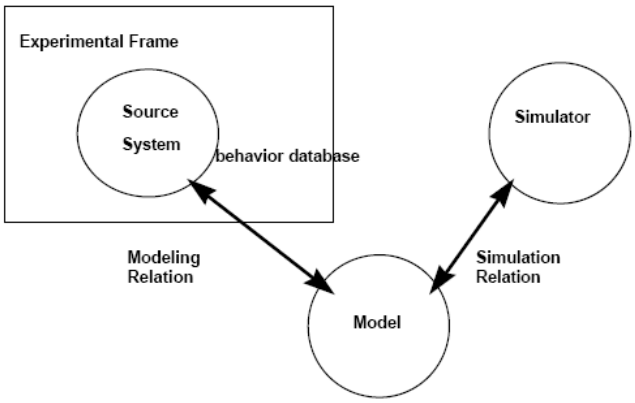


Figure Basic entities and relations

The real system: is a collection of entities (a source of data, or input/output) that interact together over time to accomplish one or more goals.

Model (informally): is a set of instructions for generating data comparable to that observable in the real system. The structure of the model is its set of instructions. The behavior of the model is the set of all possible input/output data that can be generated by executing the model instructions. Simulator: exercises the model's instructions to actually generate its behavior.

Experimental frame is a specification of the conditions under which the system is observed or experimented with. An experimental frame is the operational formulation of the objectives that motivate an M&S project. A frame is realized as a system that interacts with the system of interest to obtain the data of interest under specified conditions.

The basic objects are related by two relations:

- Modeling relation linking real system and model. It defines how well the model represents the system or entity being modeled. In general terms a model can be

considered valid if the data generated by the model agrees with the data produced by the real system in an experimental frame of interest.
- Simulation relation, linking model and simulator, represents how faithfully the simulator is able to carry out the instructions of the model.

3.1.1 DEVS modeling formalism

The DEVS formalism provides a comprehensive framework for modelling and simulation based on systems theory. The Classic DEVS System Specification was introduced by Zeigler[12]in 1976. After some 15 years a revision called Parallel DEVS (PDEVS) was introduced by Chow [12]. Both specifications define two basic system types for modelling called atomic DEVS and coupled DEVS. The dynamic behavior is specified in atomic DEVS. Both basic system types can be composed in coupled DEVS to define systems in a modular, hierarchical manner. Furthermore each specification comes along with its own simulation algorithms used for executing an entire DEVS model. A Classic or Parallel DEVS model is executed by assigning a simulator to each atomic DEVS system and a coordinator to each coupled DEVS system. Simulators and coordinators communicate by messages. A special root coordinator starts and stops a simulation run.

3.1.1.1 Classic DEVS atomic model

The classic atomic DEVS model is a structure describing the different aspects of the discrete-event behavior of a system:

Where:
- $X = \{(p, v) \mid p \in IPorts, v \in Xp\}$ is the set of input events, where IPorts represents the set of input ports and Xp represents the set of values for the input ports;
- S*:* is a set of sequential**states**
- $Y = \{(p, v) \mid p \in OPorts, v \in Yp\}$ is the set of output events, where OPorts represents the set of output ports and Yp represents the set of values for the output ports;
- is the internal transition function
- $\delta ext: Q \times X \rightarrow S$ is the external state transition function,
  With $Q = \{(s, e)/s \in S, e \in [0, ta(s)]\}$ and e is the elapsed time since the last state transition;
- is the **output function; and**
- $ta: S \rightarrow R_0^+ \cup \infty$ is the time advance function.

At any given moment, a DEVS model is in a state $s \in S$. In the absence of external events, it remains in that state for a lifetime defined by ta(s). When ta(s) expires, the model outputs the value λ(s) through a port $y \in Y$, and it then changes to a new state given by δint(s). A transition that occurs due to the consumption of time indicated by ta(s) is called an internal transition. On the other hand, an external transition occurs due to the reception of an external event. In this case, the external transition function determines the new state, given by δext (s, e, x), where s is

the current state, e is the time elapsed since the last transition, and x ∈ X is the external event that has been received.

The time advance function can take any real value between 0 and ∞. A state for which ta(s) = 0 is called a transient or intermediate state (which will trigger an instantaneous internal transition). In contrast, if ta(s) = ∞, then s is said to be a passive or infinite state, in which the system will remain perpetually unless an external event is received (can be used as a termination condition).

3.1.1.2 Classic DEVS coupled model

In the DEVS formalism, one can distinguish an atomic model from which larger ones are built and how they are connected together in a hierarchical fashion.
Atomic models may be coupled in the DEVS formalism to form a coupled model. A coupled model tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to hierarchical construction. Figure 5 shows an example of a coupled model.

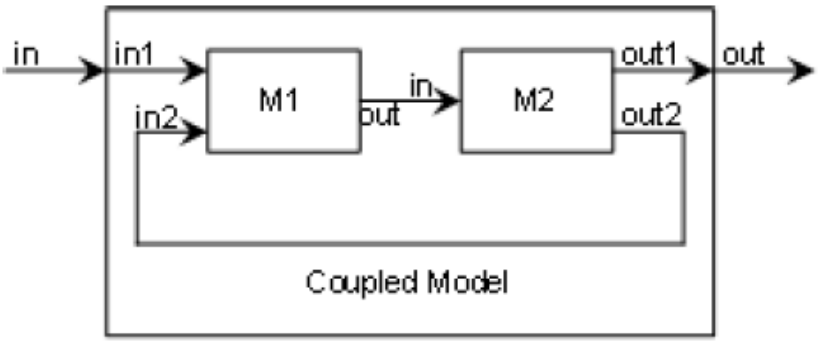


Figure Coupled DEVS model

A coupled model is formally defined by

$$CM = \langle X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC, select \rangle$$

Where

X and Y are defined as previously;
D is the set of the component names and for each d ∈ D;
M_d is a DEVS (i.e., atomic or coupled) model;
EIC is the set of external input couplings (i.e., how the inputs of the coupled model are linked to the inputs of its sub-components)
EOC is the set of external output couplings (i.e., how the outputs of the coupled model are linked to the outputs of its sub-components)
IC is the set of internal couplings (i.e., how the outputs of any sub-components are linked to the inputs of other sub-components), and
Select: $2^D \rightarrow D$ is the tiebreaker function

Coupled models group several DEVS into a composite model that can be regarded, due to the closure under coupling property, as a new DEVS model. This closure property guarantees that

the coupling of several class instances results in a model of the same class, allowing hierarchical construction.

Because multiple subcomponents can be scheduled for an internal transition at the same time, ambiguity could arise because it is not clear which transition this second component should execute first. There are two alternatives for this:

- To execute the external transition first and then the internal transition, with e = ta(s);
- To execute the internal transition first, followed by the external transition, with e = 0.

The select function provides a simple way to solve this ambiguity. The function defines an ordering over all the components of the coupled model so that only the first model to execute in the case of simultaneous internal events can be chosen.

3.1.1.3 PDEVS atomic model

In section 3.1.1.2 above, we saw that whenever two models are scheduled for state transitions at the same time, a DEVS coupled model will pick the one specified by the select function to execute first. This tie-breaking strategy is rigid. The select function introduces serialization in the execution of components when many interconnected atomic models are imminent (which could be considered as parallel in a multiprocessor environment).

Parallel DEVS (or PDEVS) is a variant to DEVS that provides a more flexible way of dealing with these ambiguities. Atomic models provide an additional confluent function to specify collision behavior for events that might be scheduled simultaneously and a mechanism for receiving multiple external events at the same time and processing them together.

An atomic PDEVS model is defined as:

Where

X, Y and S are defined as previously;
$\delta ext: Q \times X^b \rightarrow S$ is the external transition function;
$\delta int: S \rightarrow S$ is the internal state transition function;
$\delta con: S \times X^b \rightarrow S$ is the confluent transition function;
$\lambda: S \rightarrow Y^b$ is the output function; and
$ta: S \rightarrow R_0^+ \cup \infty$ is the time advance function.

PDEVS models use bags (multisets) of events for receiving inputs and collecting outputs (X^b and Y^b) instead of a single event. This allows multiple events to be processed simultaneously. Because external input events received by the component are added to the bag, external transition functions can combine the functionality of a number of external transitions into a single one, and simultaneous events (like the departure of a vehicle and a collision in the intersection) can be treated simultaneously. Also, PDEVS allows a better way to deal with collisions: the model specification includes a confluent transition function (δcon). When a collision between the internal and external functions occurs, the confluent function determines the new state of the model.

The semantics of PDEVS for internal/external transition functions is similar to DEVS. If one or more external events $X^b = \{x_1 \ldots x_n/x_i\ X\}$ occur before ta(s) expires (i.e., while the system is in total state (s, e) with e < ta(s)), the new state will be given by the model's external transition function, $\delta ext(s, e, X^b)$. If the external events Xb are received when e = ta(s), the new state of the model will be given by the confluent function (δcon). If multiple components in a coupled model are imminent, all their outputs are first collected and mapped to their influences in parallel. Then the corresponding transition function is executed for every model.

3.1.1.4 PDEVS coupledmodel

In PDEVS (Parallel DEVS), coupled models are defined as in DEVS, without the need for a select function.

Formally, a coupled model is defined as

$$CM = \langle X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC \rangle$$

Where definitions for the set of input and output events (X and Y), components (D and M_d), and couplings (EIC, EOC, and IC) follow the specifications of DEVS coupled models presented earlier in this chapter.

3.1.2. DEVS toolkits

DEVS formalism exists in many implementations. Below a non-exhaustive list [13]of tools which enable modelling and simulation based on the DEVS formalism.

- DEVSJAVA: Modeling and Simulation environment for developing DEVS-based models. The software is written in Java and supports parallel execution on a uni-processor. It supports higher-level, application specific modeling. Developed by HessamSarjoughian (Arizona State University, U.S.A.) and Bernard Zeigler (University of Arizona, U.S.A.).
- DEVS/C++[14]: based on the parallel DEVS formalism, it is a modular hierarchical discrete event simulation environment implemented in the object-oriented C++ language. Developed by Hyup J. Cho and Young K. Cho (University of Arizona, U.S.A.).
- DEVS/CD++[15][16]: is a modeling and simulation toolkit capable of executing DEVS and Cell-DEVS models supposed complex. The figure 8 below presents the CD++ in his distributed and interoperable version. It shows a master CD++ simulator coordinating two CD++ slave simulators and another non-CD++ DEVS simulator where all of the communication is performed via exchanging XML messages (over SOAP/HTTP). DEVS/CD++ is Developed by Gabriel Wainer and his students (Carleton University, Canada; Universidad de Buenos Aires, Argentina).
- DEVS/HLA[17][18]: The emergence of High Level Architecture (HLA) and the presence of Discrete Event System Specification theory are expected to provide the foundation for a capable distributed/parallel modeling and simulation environment. HLA is a DoD sponsored effort to establish a common technical framework facilitating the interoperability of all types of models and simulations among themselves and with C4I (Command, Control, Communications, Computers, and Intelligence) systems. DEVS has a theoretical foundation which makes it in principle independent of various programming languages and hardware platforms. Developed by HessamSarjoughian (Arizona State University, U.S.A.) and Bernard Zeigler (University of Arizona, U.S.A.).

- ADEVS:is a C++ library for developing discrete event simulations based on the Parallel DEVS and DSDEVS formalisms. It includes support for standard, sequential simulation and conservative, parallel simulation on shared memory machines with POSIX threads. Developed by Jim Nutaro (University of Arizona, U.S.A.).
- PyDEVS: uses ATOM3-DEVS, ATOM3-DEVS is a tool for constructing DEVS models and generating Python code for the PyDEVS simulator by Jean-Sebastien Bolduc, developed in ATOM3.ATOM3 is a tool for multi-paradigm modeling, by Juan De Lara (Autonomous University of Madrid, UAM, Spain) and Hans Vangheluwe (Mc. Gill University, Canada).
- DEVSim++: is an environment for Object-Oriented Modeling of Discrete Event Systems. Developed by Tag Gon Kim (KAIST, Korea).
- VLE (Virtual Laboratory Environment): is a multi-modeling and simulation platform. It is a powerful modeler and simulator supporting the use of different formalisms for models specification and simulation. VLE is currently using the Swing technology to enable the construction of models in several languages (Java, Python, C#).
- JAMES II (JAva-based Multipurpose Environment for Simulation II): provides support for many different formalism, among these various variants of DEVS formalisms (e.g., PDEVS). It provides a GUI, a powerful experimentation layer, and support for many more issues in M&S. It is an open architecture, distributed as open source, and it is freely reusable by anyone. Developed by Modeling & Simulation Research Group Institute of Computer Science Univsersity of Rostock.
- SimStudio_1.1: is a simulator, implemented in Java programming language that manages the communications between the models, manage time and uses the specifications defined by the modeller. It gives the modeller an opportunity to simulate his model using either the Classic DEVS or the Parallel DEVS formalisms. However, a DEVS model is either atomic or coupled. Developed by Prof. Mamadou Kaba Traoré and his students (African University of Science and Technology, Nigeria).

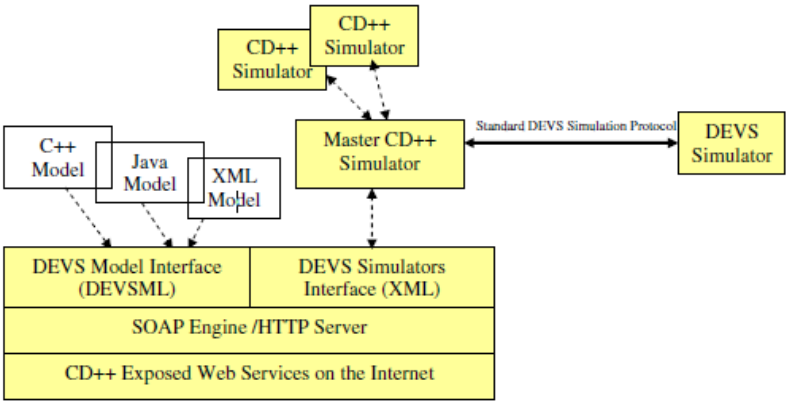


Figure Framework of Interfacing CD++ according to DEVS standard protocol over SOA

3.2Web services

Web services were designed to solve the problem of integration of heterogeneous sources and make heterogeneous systems interoperable.

Several technologies used in the recent years could have been classified as web service technology, but were not. The major difference between these technologies and web service is their standardization. Web service is based on standardized XML(as opposed to a proprietary binary standard) and supported globally by most major technology firms. This chapter will present these previous technologies so called web service and the new technology.

3.2.1 Service-Oriented Architectures (SOA)

A service-oriented architecture defines a distributed system wherein agents, known as services, coordinate by sending messages. Quoting [19]:

> *An SOA is a specific type of distributed system in which the agents are "services." [A] service is a software agent that performs some well-defined operation (i.e., "provides a service") and can be invoked outside of the context of a larger application. That is, while a service might be implemented by exposing a feature of a larger application … the users of that server need be concerned only with the interface description of the service. "[S]ervices" have a network-addressable interface and communicate via standard protocols and data formats.*

Any service-oriented architecture[20] contains three roles: a service requestor, a service provider, and a service registry:

- **A service provider** is responsible for creating a service description, publishing that service description to one or more service registries, and receiving Web service invocation messages from one or more service requestors. A service provider, then, can be any company that hosts a Web service made available on some network. You can think of a service provider as the "server side" of a client-server relationship between the service requestor and the service provider.
- **A service requestor** is responsible for finding a service description published to one or more service registries and is responsible for using service descriptions to bind to or invoke Web services hosted by service providers. Any consumer of a Web service can be considered a service requestor. You can think of a service requestor as the "client side" of a client-server relationship between the service requestor and the service provider.
- **The service registry** is responsible for advertising Web service descriptions published to it by service providers and for allowing service requestors to search the collection of service descriptions contained within the service registry. The service registry role is simple: be a match-maker between service requestor and service provider. Once the service registry makes the match, it is no longer needed in the picture; the rest of the interaction is directly between the service requestor and the service provider for the Web service invocation.

Each of these roles can be played by any program or network node. In some circumstances, a single program might fulfill multiple roles; for example, a program can be a service provider, providing a Web service to downstream consumers as well as a service requestor, itself consuming Web services provided by others.

An SOA [20] also includes three operations: publish, find, and bind. These operations define the contracts between the SOA roles:

- **The publish operation** is an act of service registration or service advertisement. It acts as the contract between the service registry and the service provider. When a service provider publishes its Web service description to a service registry, it is advertising the details of that Web service to a community of service requestors. The actual details of the publish API depend on how the service registry is implemented. In certain simple or "direct publish" scenarios, the service registry role is played by the network itself, with publish being simply an act of moving the service description into a Web application server's directory structure. Other services registry implementations, such as UDDI, define a very sophisticated implementation of the publish operation.
- **The find operation** is the logical dual of the publish operation. The find operation is the contract between a service requestor and a service registry. With the find operation, the service requestor states search criteria, such as type of service, various other aspects of the service such as quality of service guarantees, and so on. The service registry matches the find criteria against its collection of published Web service descriptions. The result of the find operation is a list of service descriptions that match the find criteria. Of course, the sophistication of the find operation varies with the implementation of the service registry role. Simple service registries can provide a find operation with nothing more sophisticated than an unparameterized HTTP GET. This means the find operation always returns all Web services published to the service registry and it is the service requestor's job to figure out which Web service description matches its needs. UDDI, of course, provides extremely powerful find capabilities.
- **The bind operation** embodies the client-server relationship between the service requestor and the service provider. The bind operation can be quite sophisticated and dynamic, such as on-the-fly generation of a client-side proxy based on the service description used to invoke the Web service; or it can be a very static model, where a developer hand-codes the way a client application invokes a Web service.

Figure 7 shows an SOA describing the relation between roles and operations.

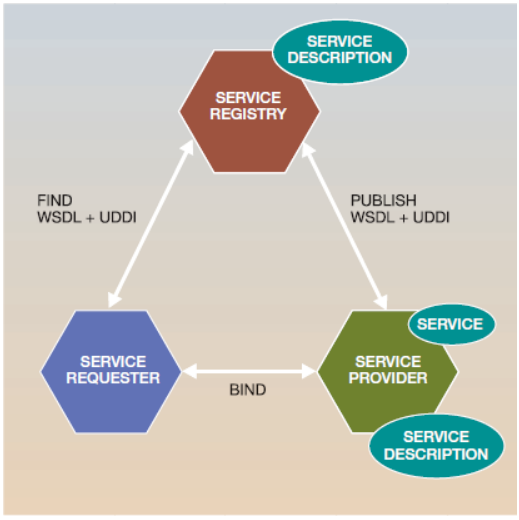


Figure Service-oriented architecture

Figure 8 illustrates a basic service-oriented architecture[22]. It shows a service consumer at the right sending a service request message to a service provider at the left. The service provider

returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and service provider. A service provider can also be a service consumer.

Figure Basic service-oriented architecture

3.2.2 Existent technologies-based SOA

In the late '90s, a number of technologies allowing remote procedure calls (RPC) from one system to another became popular.

3.2.2.1 CORBA

CORBA (Common Object Request Broker Architecture) was designed in the early 1990s to provide a mechanism for building client/server applications in heterogeneous environments.Some of the key features of CORBA are:

- **Language Neutral**: CORBA was designed to work with any language. In order to bridge the gap between differing languages, an Interface Definition Language (IDL) is used to detail the structure of all objects that will be passed along the wire into a language-neutral format. Developers then take the IDL and run it through some form of code generator for the language used on each end of the transaction to get the corresponding language-specific stub or skeleton. By doing this, it's possible to write a Visual Basic client that talks to a Java server, using CORBA as the communication layer.
- **Multiple Vendors**: Multiple vendors provide Object Request Brokers (ORBs). This allows users to pick and choose between vendors for the capabilities and costs that are right for them. Some ORB vendors only support certain languages as well.

CORBA requires the use of special ports on which the ORBs communicate and transfer the data. This present some risk for possible intrusion and attack by hackers. Also in most cases, data transferred in CORBA-based systems is sent across the wire in clear text. This makes it rather easy for hackers to listen in on communications and steal data.

As a result, compared [23]to Web services, CORBA solutions

- Are nearly as capable for cross-platform and cross-language development.
- Are harder to understand because CORBA relies on IDL to translate data; Web services use XML, which is much more human readable. Most toolsets also create the WSDL for the developer.
- Can handle higher transaction loads because they keep a persistent connection between clients and servers at the expense of servicing fewer clients per server.
- Are a much more mature technology, and many of the initial interoperability issues between vendors have been worked out already. A lot more information is currently available on CORBA as well.

3.2.2.2 JAVA RMI

Similar to CORBA, Remote Method Invocation (RMI) is built on top of the stub/skeleton architecture. RMI is the Java-specific mechanism for performing client/server calls. It is actually very similar to CORBA in many respects. The biggest difference is that because RMI is usually used for Java-to-Java architectures, there is no need for the IDL. Developers are working with true Java objects at all times. This has changed over the years with the addition of RMI communicating over Inter-ORB Protocol (IIOP, the same protocol that CORBA uses), allowing RMI to talk directly to CORBA.

Again, similar to CORBA, RMI requires that a specific port be opened for communicationsbetween the client and server. As with CORBA, this can sometimes be difficult toget network administrators to open due to security concerns.

Compared[23] to Web services, RMI is the better choice if both ends are Java based, but useless in non-Java guaranteed situations.
As a result, compared to Web services, RMI solutions

- Lock you into a purely Java solution on both the client and server
- Can be somewhat more difficult to deploy because of network port considerations
- Can handle higher transaction loads because RMI keeps a persistent connection between clients and servers at the expense of servicing fewer clients per server

3.2.2.3 DCOM

DCOM (Distributed Common Object Model) is the Microsoft mechanism for performing remote calls. Objects are again converted into a wire-friendly format and converted back to language-specific representations at the endpoints of the communication.

Although DCOM can be built in several different languages (Visual C++, Visual Basic, C#, …), it only works on Microsoft platforms. As a result, if your business does not use Microsoft servers, DCOM doesn't help you. Both ends of the transaction (client and server) need to be Microsoft systems in order to use DCOM.

Although DCOM is supported by multiple languages, the strong ties to Microsoft mean that Web services still hold an edge in flexibility. Web services can be implemented with tools from many different vendors on various platforms.

Compared[23]to Web services, DCOM solutions are nearly as capable for cross-language development, but usually lock you into a Microsoft-everywhere framework.

3.2.3Web Services

Web service is a way of communication that allows interoperability between different applications on different platforms, for example, a java based application on Windows can communicate with a .Net based one on Linux. The communication can be done through a set of XML messages over HTTP protocol.

At the beginning, the Internet is developed for *human-to-Machine* interaction as shown in Figure 9. Human at the one end looks at whatever information that is on the web through web browsers.

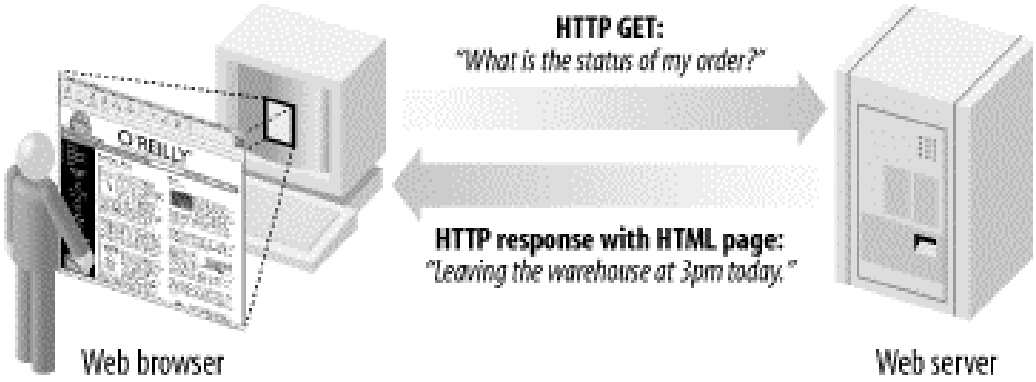


Figure human-to-Machine interaction

With web services [24], we move from a human-centric Web to an application-centric Web (See Figure 10). This does not mean that humans are entirely out the picture! It just means that conversations can take place directly between applications as easily as between web browsers and servers.

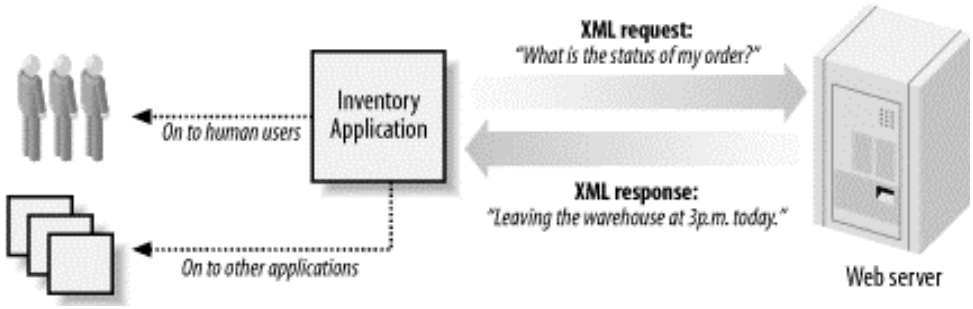


Figure The application-centric Web

The World Wide Web Consortium (W3C) defines a Web service as[19]:

> *a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards*.

Given this definition, a machine to machine interaction using web service can be represented as shown in Figure11 below:

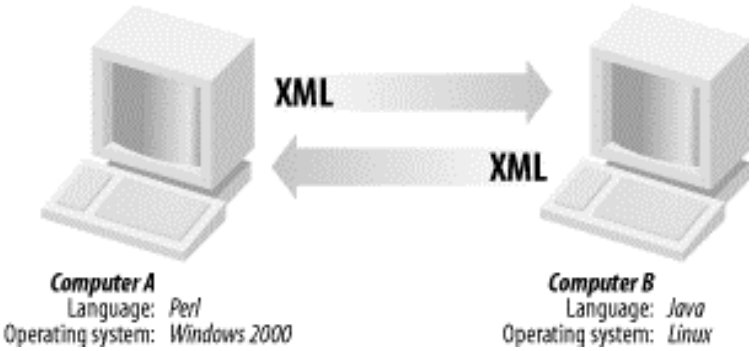


Figure A basic web service

Full details of Web Services are discussed in the next section of the chapter.

3.2.3.1 Web Services components

UDDI, WSDL, XML messaging (XML-RPC, SOAP), and HTTP form what is commonly referred as Web Services. Figure 12 presents the Web services protocol stack [24].

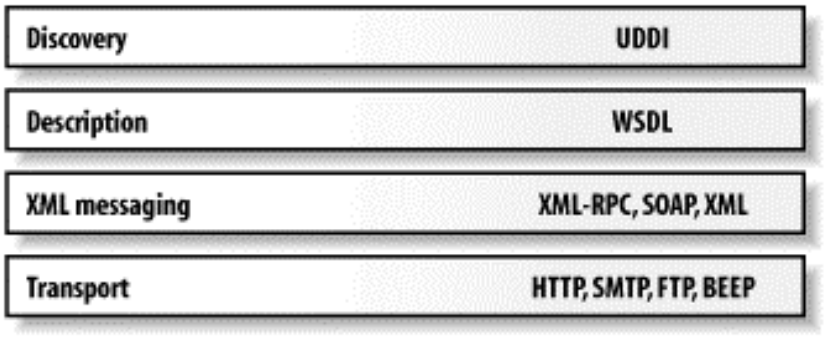


Figure Web service protocol stack

XML messaging: XML has exploded onto the computing scene in recent years. It has gained rapid acceptance because it enables diverse computer systems to share data more easily, regardless of operating system or programming language. When developers decided to build a web service messaging system, XML was therefore a natural choice. There are two main contenders for XML messaging: XML-RPC and SOAP.

XML-RPC: is a simple protocol that uses XML messages to perform Remote Procedural Calls (RPCs). Requests are encoded in XML and sent via HTTP POST. XML responses are embedded in the body of the HTTP response. Because XML-RPC is platform-independent, it allows diverse applications to communicate. For example, a Java client can speak XML-RPC to a Perl server.XML-RPC is the easiest way to get started with web services. In many ways, it is simpler than SOAP and easier to adopt. However, unlike SOAP, XML-RPC has no corresponding service description grammar. This prevents automatic invocation of XML-RPC services - a key ingredient for enabling just-in-time application integration.

- **SOAP:**SOAP is an XML-based protocol for exchanging information between computers. Although SOAP can be used in a variety of messaging systems and can be delivered via a variety of transport protocols, the initial focus of SOAP is remote procedure calls

transported via HTTP. SOAP therefore enables client applications to easily connect to remote services and invoke remote methods. For example, a client application can immediately add language translation to its feature set by locating the correct SOAP service and invoking the correct method.

- **SOAP Message:** With the stabilization of the Internet, SOAP a new approach based on machine-to-machine interaction is adopted. SOAP messages are XML documents that are embedded in the transport's request and response. SOAP messages can be transported over HTTP for the runtime invocation.From the viewpoint of the Internet layer, computer A is sending an XML document to computer B via HTTP (See Figure 13). Computer B's firewall policy states that XML documents are allowed to pass through via HTTP. The Web server on computer B receives the file and hands it to the SOAP processor, which uses an XML parser to read the document.From the XML parser's point of view, the document is simply a well-formed and valid XML document. The SOAP processing engine evaluates the file against the rules of the SOAP grammar and an XML schema. It examines the SOAP vocabulary to determine if it is valid. If it is valid, the SOAP processor makes a call to the Web service described in the SOAP document and passes it any parameters that the document might contain.When the Web service finishes its processing, it creates a response that is formatted in an application-specific way. It wraps this response in a SOAP message format, which is a valid XML document also. It stores this document in a file and hands it to the Web server for delivery back to the client computer, computer A.On computer A, the HTTP client program receives the response file. It calls the SOAP processor to parse and to validate it. If it is a valid document, the SOAP processor passes the response back to the Web services client program that sent the original request.In some cases the SOAP message might not be a method call and response; it could simply be a single call or even just a document being moved.
- **Type of SOAP message:** The SOAP standard contains two parts: the header (optional) that carries processing instructions and the body that contains the payload. The payload contains the information to send. The two types of SOAP messages are documents and Remote Procedure Calls (RPCs). The payload of a document message is any XML document that you want moved from one computer to another. An RPC is a method call that is intended to be executed on the Web service's computer. The RPC message performs the same function as an ordinary method call in an ordinary programming language. The difference is that this call can take place over the Internet.Figure 14shows the basic outline of a SOAP message. The root element is the envelope tag, which contains two more elements: body and header.
- **SOAP Envelope Tag:** A SOAP message is defined as beginning with the tag <SOAP-ENV:Envelope> and ending with the tag </SOAP-ENV:Envelope>. SOAP messages cannot be sent in batches.In SOAP 1.1, the SOAP_ENV:Envelope tag is normally constructed using the following syntax: <SOAP-ENV: Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >. The string ***xmlns*** is a keyword in XML that stands for XML namespace. The namespace is used to uniquely identify all tags in order to avoid tag name conflicts. The SOAP-ENV part is the name that SOAP requires to be used as the prefix for all the tag names that SOAP defines. The string http://schemas.xmlsoap.org/soap/envelope/ specifies the version of the SOAP that is being used. The Web service that receives the request can look at this string to determine whether it is capable of communicating using that version of SOAP. Two other

namespaces that are heavily used in SOAP are ***xsd*** and ***xsi***. The ***xsd*** namespace specifies that these tags come from the XML schema definition. The ***xsi*** namespace indicates that these tags come from the XML schema–instance definition.

- **SOAP Body Tag:** The body of the SOAP message begins with the tag <SOAP-ENV:Body> and ends with the tag </SOAP-ENV:Body>. This is where the payload of the SOAP message is placed. Normally, that payload is a method call to a remote computer, complete with parameter values. Sometimes, however, it is simply an XML document that is being transferred.
- **SOAP Header Tag:** The SOAP-ENV:Header element is optional in a SOAP message. If a header is present, however, it must be the first child element that appears in the SOAP envelope. It starts with <SOAP-ENV:Header> and ends with the tag </SOAP-ENV:Header>

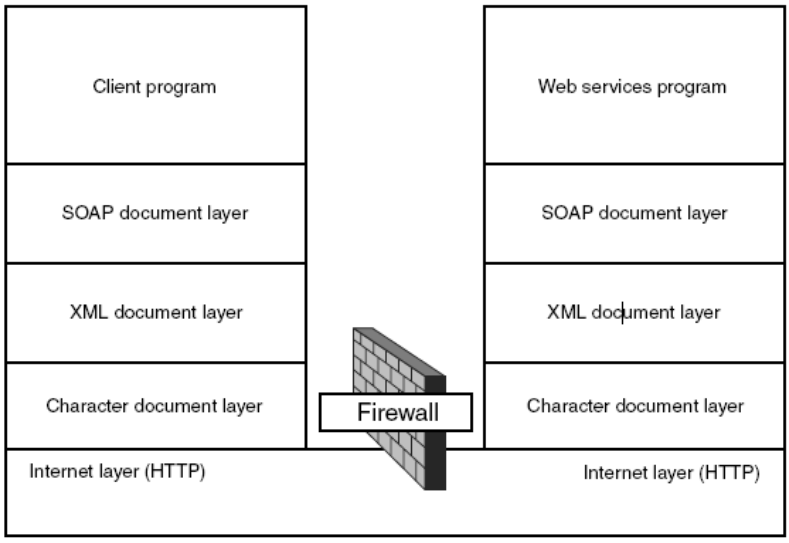


Figure SOAP message communication

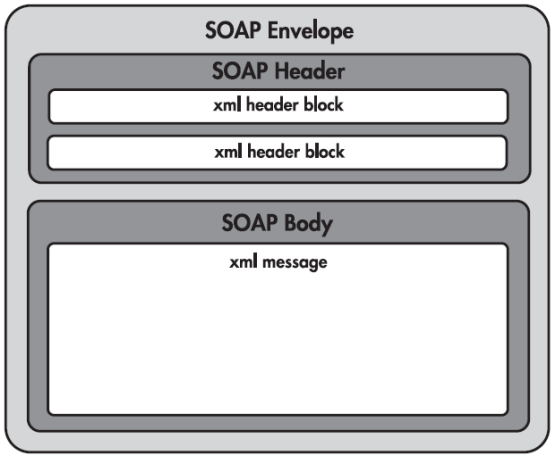


Figure Diagram of a SOAP message

3.2.3.2 Service Description: WSDL

WSDL forms the basis for Web Services. Figure 15 illustrates the use of WSDL [22]. At the left is a service provider. At the right is a service consumer. The steps involved in providing and consuming a service are:

1. A service provider describes its service using WSDL. This definition is published to a directory of services. The directory could use Universal Description, Discovery, and Integration (UDDI). Other forms of directories can also be used.
2. A service consumer issues one or more queries to the directory to locate a service and determine how to communicate with that service.
3. Part of the WSDL provided by the service provider is passed to the service consumer. This tells the service consumer what the requests and responses are for the service provider.
4. The service consumer uses the WSDL to send a request to the service provider.
5. The service provider provides the expected response to the service consumer.

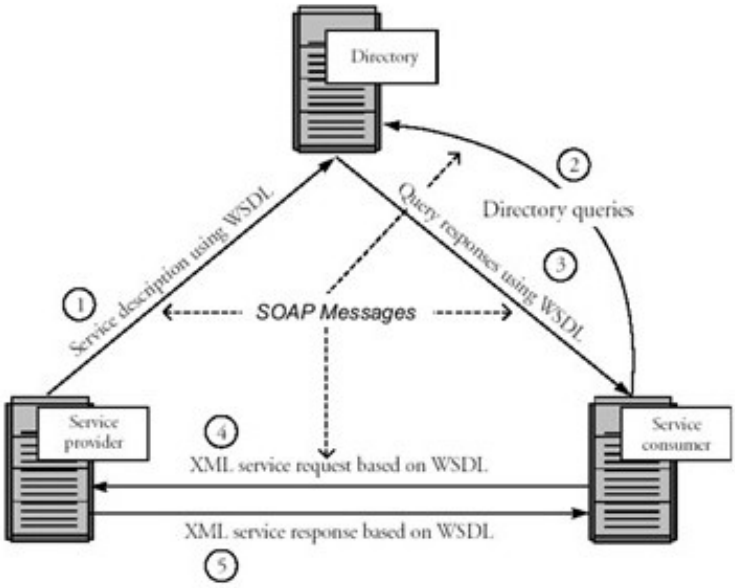


Figure Web Services basics

3.2.3.3 Service Discovery (UDDI)

UDDI [26] provides a mechanism for clients to find Web services. Web services are meaningful only if potential users may find information sufficient to permit their execution. The focus of Universal Description Discovery & Integration (UDDI) is the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other Web services providers, (2) the Web services they make available, and (3) the technical interfaces which are used to access those services. Based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP, UDDI provides an interoperable, foundational infrastructure for a Web services-based software environment for both publicly available services and services only exposed internally within an organization.

3.2.3.4 Why Web Services

Web services represent a new architecture for creating applications that can be accessed from a different computer. Normally, a Web service is identified by a URL, just like any other Web site. The Web services architecture is based on sending XML messages in a specific SOAP format. XML can be represented as plain ASCII characters, which can be transferred easily from computer to computer. The implications of this are significant:

- It doesn't matter what kind of computer sends the SOAP message or on what operating system it is running.
- It doesn't matter where in the world the client is sending the message from.
- It doesn't matter what language the software that the client is running on was written in.
- There is no need for the client to know what type of SOAP processor is running on the server.

In [25], web servicesare categorized in three (3) domains:

- **Business to Business (B2B) Services**
  - Application and data integration
  - Cost-savings
- **Business to Consumers (B2C) Services:**
  - Code-reuse
  - Cost-savings
- **Device to Device (D2D) Services:**
  - Platform independent (Hardware, OS and Programming language)
  - Cost-savings

3.2.3.5 Web services Tools and Vendors

Web services tools[23] are numerous because so many software vendors have bought into its promise. In addition, a few startups have also written Web services development tools. The result is a wide variety of choices. You not only get to choose between vendors and IDEs, but you also get to determine what level of tool you want to employ.The following is a list of some existing tools:

- **Apache CXF or Axis2:** Apache Software Foundation coordinates the creation of open source projects. One of its projects is a SOAP engine that is normally used with its Tomcat server.

  Apache CXF: Apache CXF is the product of two projects, Celtix and XFire, hence the name CXF.Apache CXF is an open source web service framework that provides an easy to use, standard-based programming model for developing web services. Web services can be implemented using different application protocols like SOAP, XML, JSON, RESTful HTTP, and support various transport protocols like HTTP or JMS (Java Message Service).

  Apache Axis2: Apache Axis2 is the next generation web service framework from Apache. The Apache software foundation started Apache SOAP as its first web service framework. Next, they developed Apache Axis, which became one of the very successful projects at Apache and is still used heavily in the industry. Due to rapid changes in the industry and demands from the user community, Apache Axis alone was not able to fulfill

those requirements, thus the Apache Web Service community initiated Apache Axis2 project in 2004. In a short period of time, Apache Axis2 has become the de facto open source Java Web Service framework, which is now heavily used in both the industry and in academia.

Axis2 use various standards like JAX-WS, JAXRS, JAXB, SOAP, WSDL, REST, MTOM, WS-Security, WS-Policy, XML Encryption and XML Signature.

- **Java:** Sun Microsystems has created a set of optional packages that can access UDDI registries, generate WSDLs, and so on.
  The Java Web Services Developer Pack (JWSDP) is a toolkit provided by Sun to demonstrate how to build Web services solutions using Java. The toolkit is meant for teaching purposes, not production use.
  The Java Web Services Developer Pack can be downloaded from the Sun Java Web site.
- **Visual Studio .NET**: Microsoft's new way to create Web services is to use this product in conjunction with any one of the Visual Studio languages such as Visual Basic, Visual C++, or C#.
- **Web Services .NET Clients**: Clients created using .NET that can interact with any Web service.
- **BEA WebLogic Workshop**: BEA is a leading J2EE vendor that has created a user-friendly way to create Web services by using an elaborate IDE.
- **IBM WebSphere Studio Application Developer (WSAD):** IBM has made the creation of Web services a part of its comprehensive package called WSAD.
- **Other Important Products**: Many lesser-known companies have quality entries in the Web services development tool market: Iona XMLbus, The Mind Electric GLUE, PocketSOAP, and SOAP::Lite.

3.2.3.6 Example of services

A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging.

- *E-service using UDDI in SELF-SERV:* Web service also called e-service [27] is a well-defined abstraction that allows users to access functionalities offered by Web applications (see Figure 16). Web services are published and discovered using some service discovery tools integrating UDDI (Universal Description, Discovery and Integration). UDDI is integrated to SELF-SERV (compoSingwEbaccessibLeinFormation&buSinesssERVices) platform for Web services advertisement and discovery. SELF-SERV is a platform for rapid composition of Web services. An elementary service is an individual Internet accessible application (e.g. a Java program) that does not reply on another Web service to fulfill user requests. A composite service aggregates multiple Web services, which are referred to as its components. A community describes the capabilities of a desired service without referring to any actual Web service providers.Any service wishing to participate in the SELFSERV platform needs to register with the Service Discovery Engine. The Service Editor provides a visual interface to create new services by composing existing Web services using statecharts. Before this procedure takes place, the service composer must find the relevant services among the advertised Web services in the Service Discovery Engine at design time.The main advantage is that Web services can register with the UDDI registry and can be discovered at the design time by service composer, to compose new Web services using SELF-SERV platform.SELF-SERV does not address

the management of the global QoS (Quality of Service) requirements. The service selection process of the approach, shown in figure above, uses a local selection strategy. This means that the selection of a service is determined independently to other services of the composite service. Even though the selection is locally optimal, this does not ensure the global QoS constraints will be satisfied.

- *Daios (Dynamic and Asynchronous Invocation of Services) framework:*Daios is a message-based service framework that supports SOA implementation, allowing dynamic invocation of SOAP/WSDL-based and RESTful services. Figure 17, Clients communicate with the framework front end using Daios messages.The framework supports the service-oriented architecture publish, find, and bind paradigm. Daios framework lets application developers create stubless and dynamic service clients that aren't strongly coupled to a specific service provider. Instead, Daios's dynamic interface offers a high degree of provider transparency that lets applications exchange service providers at runtime.

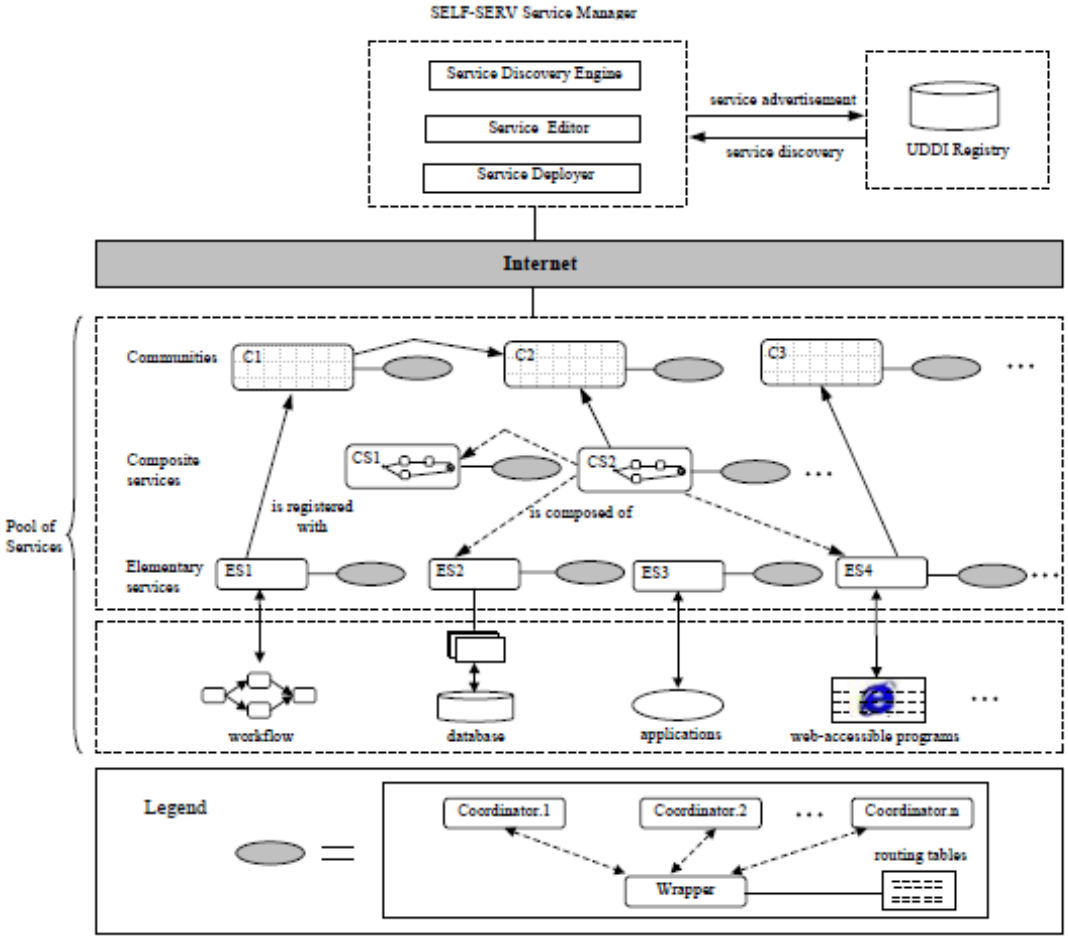


Figure SELF-SERV Architecture

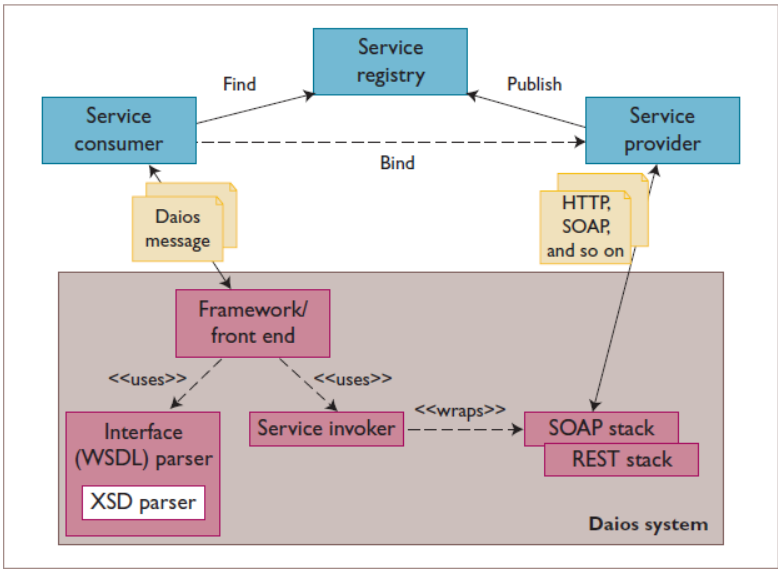


Figure The Daios framework's overall architecture.

Chapter 4: DEVS modeling and simulation as a service

4.1 Overview

The present work is inspire by the theoretical work of M. Traoré and al. [9]on DEVS Markup Language (DML).DML is a platform defined to enable the sharing and reuse of models as well as the simulation of distributed and heterogeneous models across the World Wide Web. Two (2)ways are provided to achieve this:

- **Simulator-based interoperability:** The main idea of this approach, as used in DEVS/SOA, is to have a collection of simulation services distributed over the internet (see Figure 18).
- **Model-based interoperability:** Models are deployed as web services instead of simulators. Two ways of model-based interoperability are discussed, the on-line (dynamic) model-based interoperability and the off-line (static) model-based interoperability. *In the on-line model-based interoperability*, from a model written in a given framework its representation is generated in a platform-independent language, such DEVS Markup Language (DML) or the DEVS Modeling and Language (DEVSML). *In the off-line model-based interoperability*, the DML description of existing models written for a specific framework is generated and stored in model repositories, accessible through Internet, so that anyone (with the proper authorizations) can access them. The goal is to use existing model that could fulfill the modeler's requirements. If there is none, he can write his own but still take advantage of existing works by reusing them in his coupled models. The advantage is that the modeler ends up simulating his models and those of other scientists on his own simulating platform. Compared to on-line interoperability, this approach reduces the communication with the remote servers. The drawback is that there is no synchronization version if the origin is updated.

Since DML is a platform and language-independent format for describing and sharing DEVS models, called DEVS Markup Language; it solves partially the problem of the distribution and interoperability of DEVS-based model and DEVS-based simulation discuss in this present work.

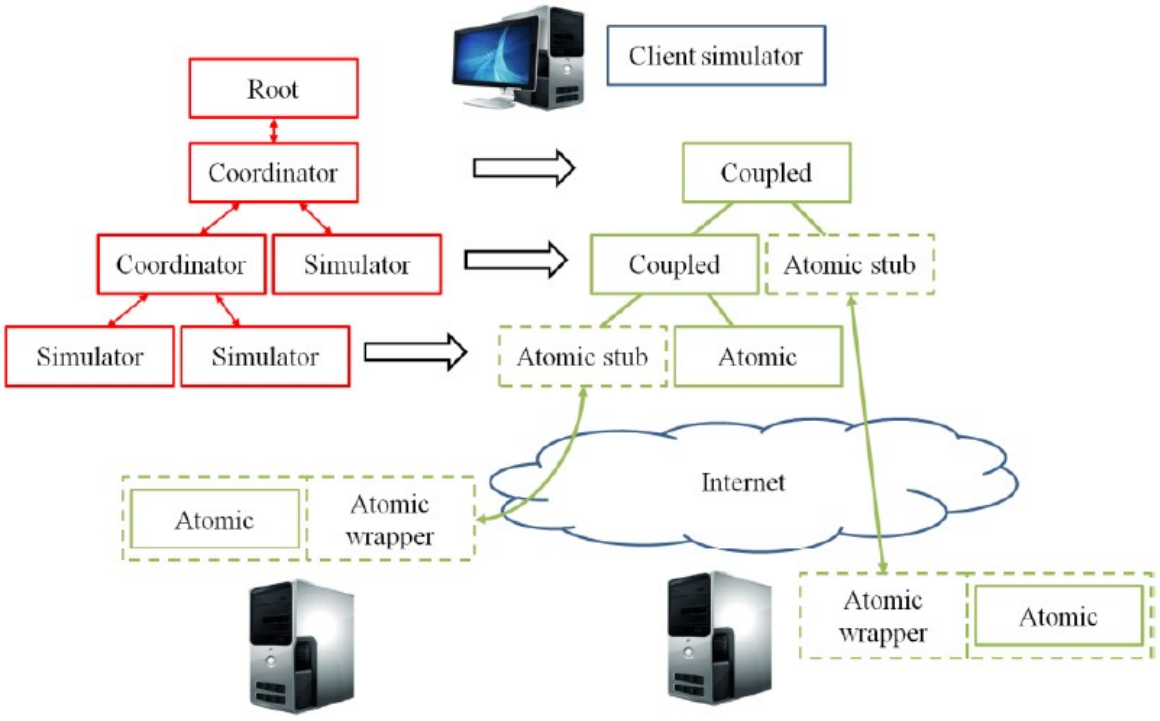


Figure Local simulation of distant and local models

4.2 DEVS modeling and simulation as a service

In this work, we have only considered a HTTP based service message transport and the service models and simulators were developed in the context of publish and subscribe.

4.2.1 Example of DEVS modeling and simulation as a service using the SimStudio_1.1

SimStudio_1.1 is an object oriented, scalable, and flexible DEVS modeling and simulation environment.The goal of this section is to deploy the SimStudio_1.1 as a service using the following tools:

- Java JDK
- Java EE Eclipse Helios
- Apache tomcat 7
- Apache Axis2
- Apache Axis2 plugins for Eclipse

4.2.1.1 Axis2 Web Services deployment of the SimStudio_1.1

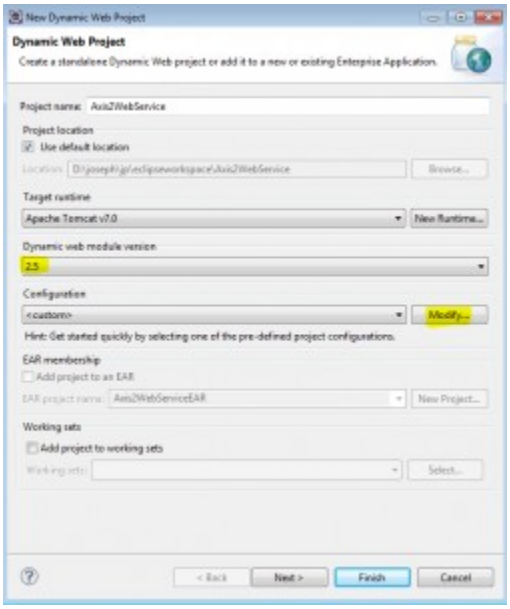
There are different Web Services implementation engines available for Java. In this part of the work, we will use Axis2 for implementing a web service.Axis is the Apache Software foundation SOAP engine designed for deploying Web Services. It's available for both Java and C++ environments. We will deploy a Java jar file as a Web service in conjunction with Eclipse wizards and Tomcat, Apache's other open source project for a J2EE-Servlet/JSP container.Eclipse

does not come with pre-configure Axis2 setup. To create a web service using Eclipse for Axis2, we need to set up the Eclipse.

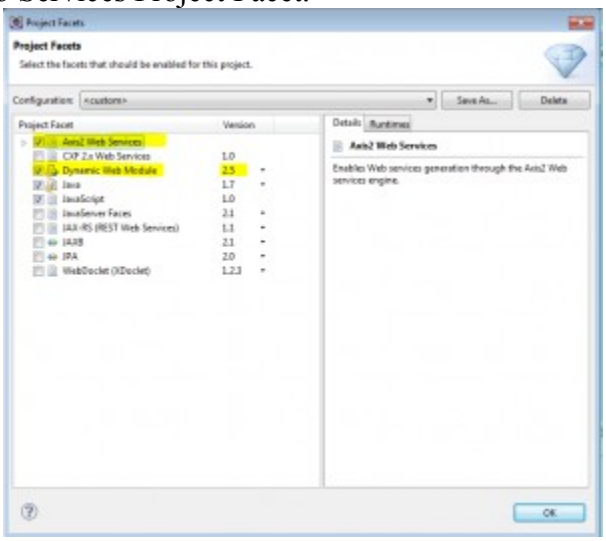
4.2.1.1.1 Configuration of Eclipse for Web Services with Axis2

We will use Eclipse Helios, a Java EE IDE for Web Developers.

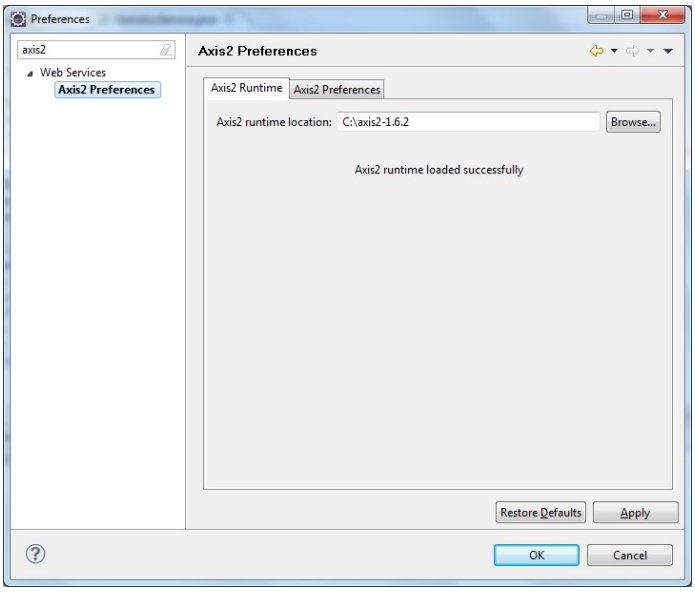
- The Dynamic Web Module version to use will be 2.5



- We need to add Axis2 Web Services in Project Facets. Click Modify button for server runtime configuration while creating the dynamic web project and select the Axis2 Web Services Project Facet.



- Configure the Axis2 binary folder downloaded from Apache Axis website in Window -> Preferences -> web service Axis2 runtime (refer image below)



- Copy the two (2) plugins folder Downloaded from Apache Axis website into the dropins folders contained in the Eclipse folder. The two (2) plugins are:
- Apache Axis2 Service Archive Generator Wizard: As a part of the Axis2 tool set, the service archive generator is an important tool that allows the generation of service archives (an "aar" file or a "jar" file) that can be deployed as a web service to Axis2.
- Apache Axis2 Code Generator Wizard: The Axis2 Code Generator Wizard is the other important tool that allows you to generate WSDL2Java and Java2WSDL code, which is the heart of developing and testing Web services.
- Developing and deploying a web service can be done using the Top-down (Contract First) and Bottom-up (Code First) approach using Axis2 Eclipse Plugins.

In this present work, we'll use the Bottom-up approach to create and deploy Web Services.

4.2.1.1.2 Configuration of services.xml

In Axis2, the service configuration file is the services.xml file. In Apache Axis2/Java, we can either deploy a single service or multiple services in a single service archive file. Irrespective of the way we deploy our service, the service archive file must contain a services.xml file if it is to be a valid service. It should be available in the META-INF directory of the archive file (.jar for this work). Depending on the way we deploy our service, the syntax of the services.xml will vary. There are two types of services.xml: one for a single service and one for a service group.

Writing services.xml for a Single Service

The root XML element of services.xml for a single service is service, so it looks like,

```
<service>
.....
</service>
```

The xml code below shows a complete example of a single service.

```
<servicename="AbstractSimulator" >
        <description>AbstractSimulator      </description>
        <messageReceivers>
                <messageReceivermep=”http://www.w3.org/2004/08/wsdl/in-only”
                class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
<messageReceivermep=”http://www.w3.org/2004/08/wsdl/in-out”
                    class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
        </messageReceivers>
        <parameter name="ServiceClass">simulator.AbstractSimulator</parameter>
</service>
```

services.xml of AbstractSimulator

name: The service name will be the name of the archive file if the .jar file contains only one service, or else the name of the service will be the name given by the name attribute.
Description: (Optional) If we want to display any description about the service via Axis2 web-admin module, then the description can be specified here.
parameters: A services.xml can have any number of top level parameters and all the specified parameters will be transformed into service properties in the corresponding AxisService. There is a compulsory parameter in services.xml called ServiceClass that specifies the Java class, which performs the above transformation. This class is loaded by the MessageReceiver.
MessageReceiver: is used to receive data from an external system. A Message receiver is MEP (Message Exchange Pattern) dependent, so we must have different message receivers for different MEPs. We can specify the Message Receiver along with the MEP that is implemented in the services.xml. Then, at deployment, the correct Message receivers will be set for the operations in the services.

Writing services.xml for a Service Group

A service group is a convenient way of deploying multiple services together in one service archive file. Of course there will be a logical relationship between the services at runtime. The only difference in the services.xml for a service group and single service is its root element. For a service group, the root element is serviceGroup, and we can have multiple service elements inside the serviceGroup element.

```
<serviceGroup>
<service name=service1>
          .....
<service>
<service name=service2>
          .....
</service>
</serviceGroup>
```

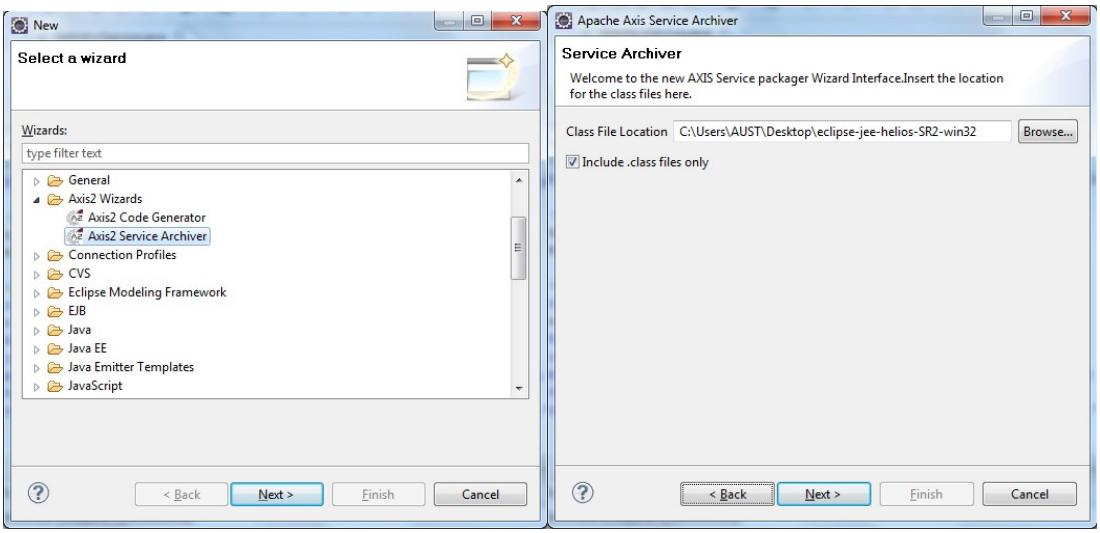
The name of the service group is the name of the service archive file. The name attribute of the service element is mandatory and the name value should be unique across the whole server. There cannot be two services with the same name.

Writing a services.xml for a service group is easy. As mentioned above, the service group is just a collection of service elements, so all the syntaxes mentioned above will be applicable. The xml code below shows the serviceGroup used in services.xml for the generation of service archive in this work. It is not completely listed because it must be composed by 31 single services relative to the number of java classes in the SimStudio_1.1.

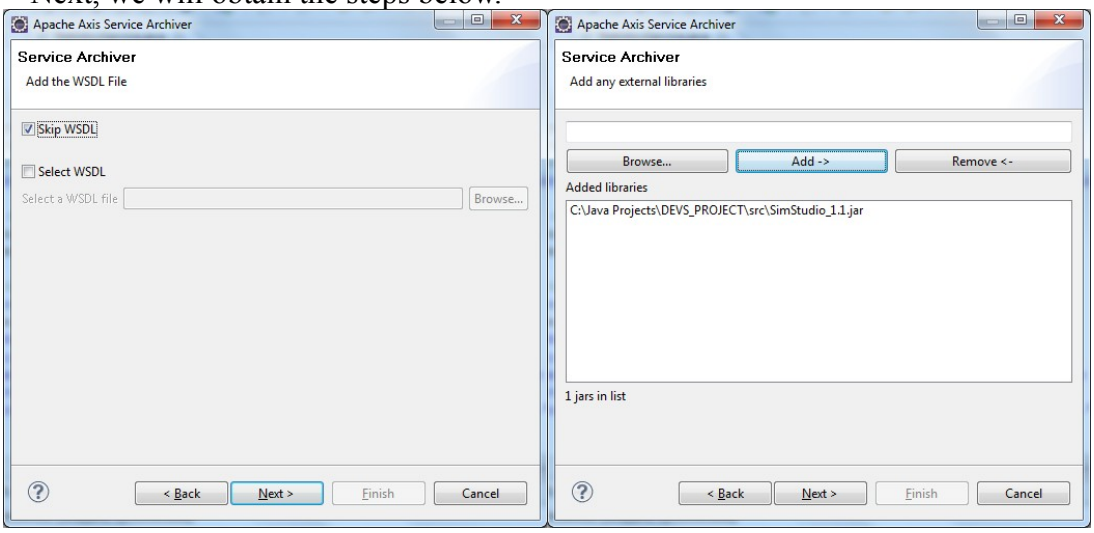
```
<serviceGroup>
        <service name="AbstractSimulator" >
                <description>AbstractSimulator</description>
                <messageReceivers>
                        <messageReceivermep=”http://www.w3.org/2004/08/wsdl/in-only”
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
                        <messageReceivermep=”http://www.w3.org/2004/08/wsdl/in-out”
                            class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
                </messageReceivers>
                <parameter name="ServiceClass">simulator.AbstractSimulator</parameter>
        </service>
        <service name="AtomicModel" >
                <description>AtomicModel</description>
                <messageReceivers>
                        <messageReceivermep=”http://www.w3.org/2004/08/wsdl/in-only”
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
                        <messageReceivermep=”http://www.w3.org/2004/08/wsdl/in-out”
                            class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
                </messageReceivers>
                <parameter name="ServiceClass">model.AtomicModel</parameter>
        </service>
        ……
        ……
</serviceGroup>
```

4.2.1.1.3 Creation and deployment of Service archive

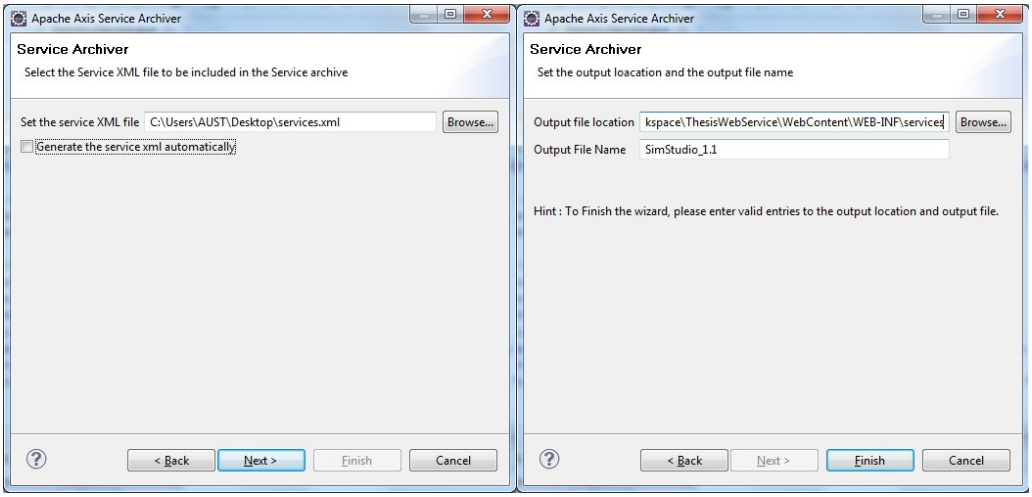
We will use the bottom up style for Web Services creation as the work will be based on an existent Java jar file (the DEVS SimStudio_1.1) to deploy as a service.To avoid having many services archive for each class contained in the SimStudio_1.1 to be deployed, we’ll use the single services.xml using serviceGroup as define above.To create the service archive to be deployed, we will use the Apache Axis2 Service archive Generator Wizard plugin. The following step will lead to the creation of the jar file service archive.
File -> new -> others -> Axis2 Wizards ->Axis2 Service archiver



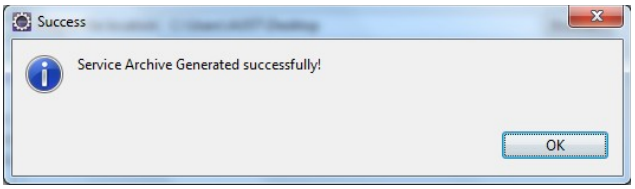
Next -> Next, we will obtain the steps below.



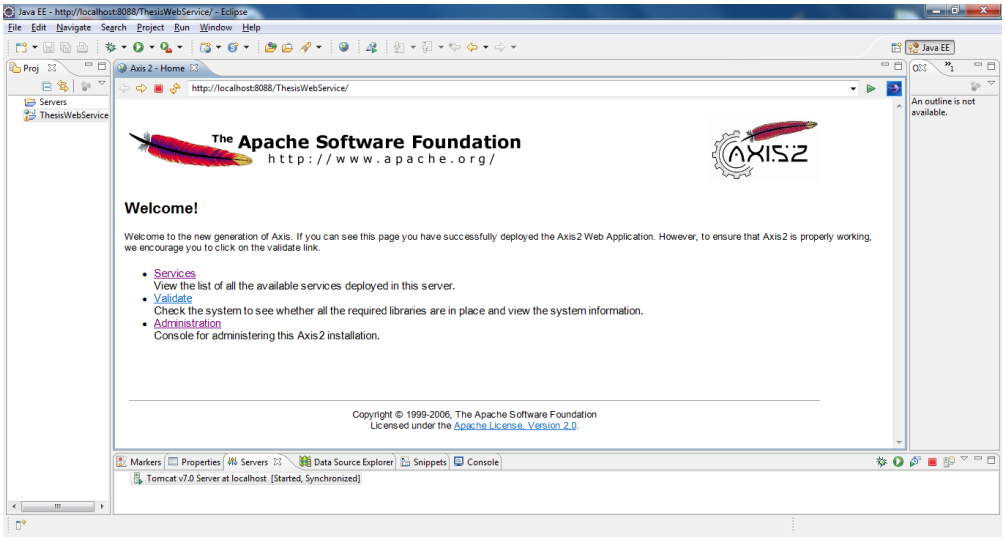
After choosing Skip WSDL and adding the location of the jar file needed, we will set the path of the service.xml previously created. Then we will indicate the path of the location to receive the service archive in creation and the name of the this service.



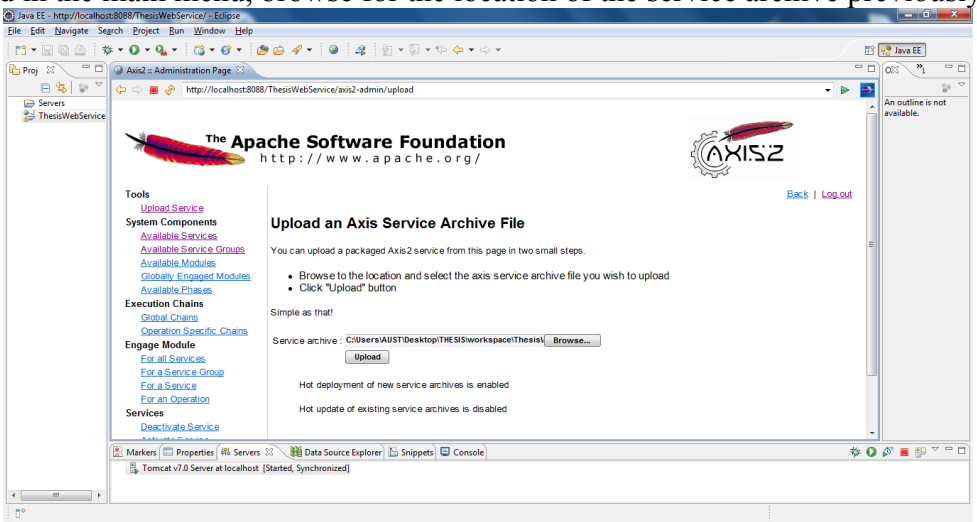
The figure below confirm the creation of the service archive.



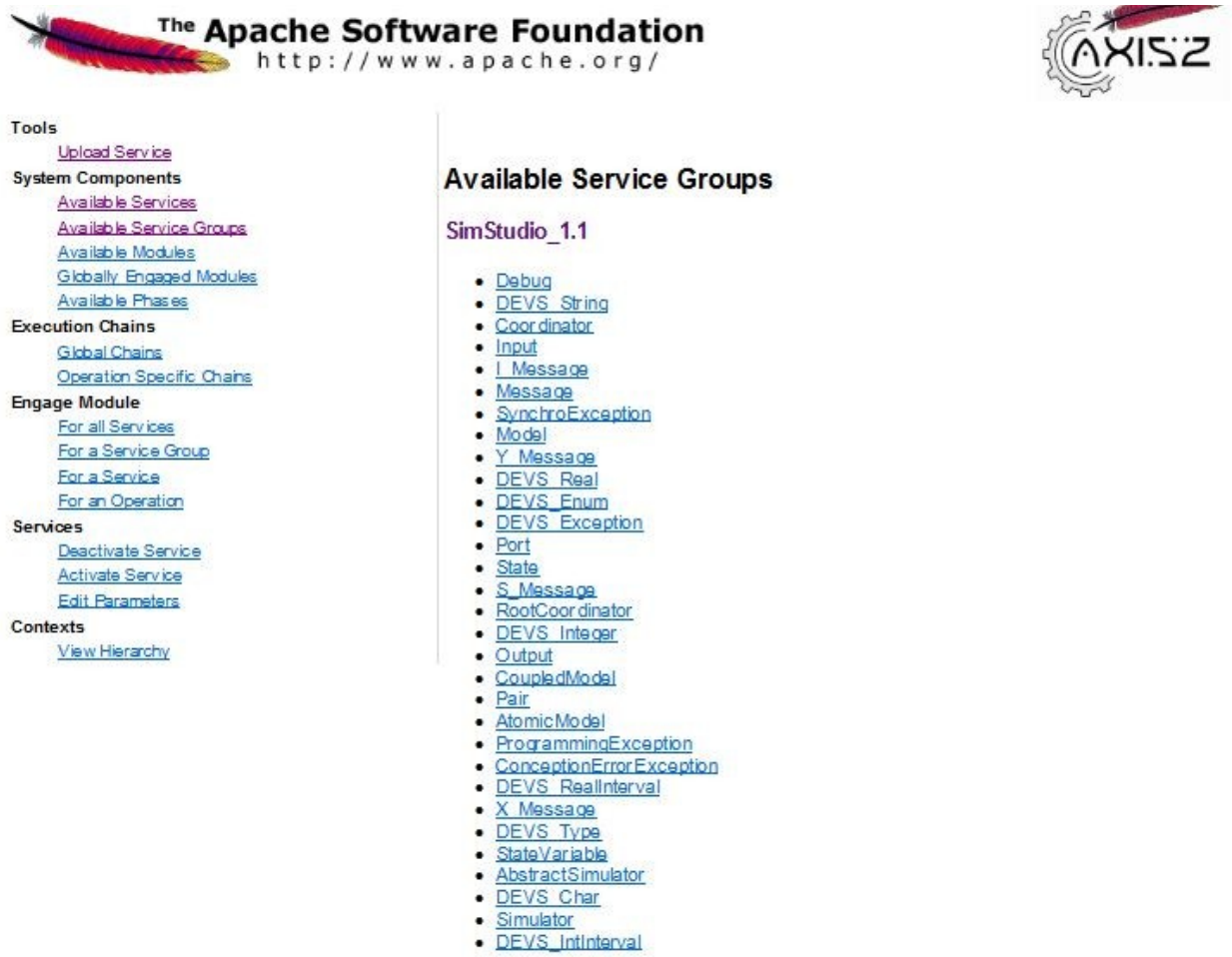
To deploy the service archive previously created, we need to run the project on server. We will obtain the Home page of Axis2. If we click on Services we'll see all the service available on the server.



In our case we need to click on Administration and upload the new service created before checking on Services if the new service created is available. This requires a Username (admin) and a Password (axis2). After user connection established, on the left tab menu click on Upload Service and in the main menu, browse for the location of the service archive previously created.



After uploading the service archive previously created, we can list the available services by clicking on "Available services or Available Services Group" on the left side of the menu. This can show the result below.



4.2.1.2Consuming the Axis2 Web Services

Apache Axis also provides a set of libraries that allow creating Web Services clients. The WSDL contract Axis created during web service deployment can be used to generate a web service client in any language or platform that supports Web Services. In other words, web services consumers can write a program in whichever language they prefer (C++, .Net, VisualBasic, PHP, …) to call the service, without even knowing the web service is written in Java. The figure below represents the WSDL of the service Input deployed in the previous section.

Writing a web service client requires normally to copy the address of the WSDL file for the web service in need to contact and feed it into a specific language-based tool, which would then give a determinate language skeleton to work with.

We have tried to consume the web service at client level but we didn't succeed. The following section will present the challenge and perspectives for the future work.

Chapter 5 Conclusion

This section presents the result, challenges we faces during our researches and some perspectives for the future work.

5.1 Result

As result, the present work helps to study more about the interoperability of DEVS-based systems using SOA-based architecture. This study ends up with the test of a web service-based DEVS modeling and simulation. The result of the test is acceptable since the deployment of the DEVS toolkit SimStudio_1.1 as a service is successful. However the test to consume the web service is not concluded.

This work also helps to have more knowledge on web services technologies and to get familiar with some of the tools and techniques used to achieve the objective.

5.2Challenges

Web services are designed to support interoperable machine-to-machine interaction over a network. There exist numerous development environmentsand technics for developing Web services. The right choice is challenging, we need to do the following:

- Choosethe Type of Web Service to Use:
    - SOAP Web Services
    - or RESTFUL Web Services
- Choose tools and technical approaches
- Get familiar with those technologies

5.3 Perspectives

- More work need to be done to achieve the Objectives
- Explore others technical possibilities other than Axis2 for deploying web services and compare to the present methodology applied in this work for advertising future programmer in web service-based DEVS modeling and solution

References

[1] T. Wutzler and H. S. Sarjoughian, “Interoperability among parallel DEVS simulators and models implemented in multiple programming languages,” *Simulation*, vol. 83, no. 6, pp. 473–490, 2007.

[2] C. Seo and B. P. Zeigler, “Interoperability between DEVS simulators using service oriented architecture and DEVS namespace,” in *Proceedings of the 2009 Spring Simulation Multiconference*, 2009, p. 157.

[3] G. A. Wainer, K. Al-Zoubi, O. Dalle, D. R. Hill, S. Mittal, J. L. R. Martín, H. Sarjoughian, L. Touraille, M. K. Traoré, and B. P. Zeigler, “15 DEVS Standardization: Foundations and Trends,” *Discrete-Event Model. Simul. Theory Appl.*, p. 389, 2010.

[4] S. Mittal, J. L. Risco-Martín, and B. P. Zeigler, “DEVSML: automating DEVS execution over SOA towards transparent simulators,” in *Proceedings of the 2007 spring simulation multiconference-Volume 2*, 2007, pp. 287–295.

[5] G. A. Wainer, K. Al-Zoubi, D. R. Hill, S. Mittal, J. L. Risco, H. S. Martín, L. Touraille, M. K. Traoré, and B. P. Zeigler, “16 An Introduction to DEVS Standardization,” *Discrete-Event Model. Simul. Theory Appl.*, p. 393, 2010.

[6] G. A. Wainer and P. J. Mosterman, *Discrete-event modeling and simulation: theory and applications*. CRC Press, 2010.

[7] A. Moreno, J. L. Risco-Martín, E. Besada, S. Mittal, and J. Aranda, “DEVS/SOA: Towards DEVS Interoperability in Distributed M&S,” in *Distributed Simulation and Real Time Applications, 2009. DS-RT’09. 13th IEEE/ACM International Symposium on*, 2009, pp. 144–153.

[8] G. A. Wainer, K. Al-Zoubi, D. R. Hill, S. Mittal, J. L. Risco, H. S. Martín, L. Touraille, M. K. Traoré, and B. P. Zeigler, “17 Standardizing DEVS Model Representation,” *Discrete-Event Model. Simul. Theory Appl.*, p. 427, 2010.

[9] L. Touraille, M. Traoré, D. R. Hill, and others, “On the Interoperability of DEVS components: On-Line vs. Off-Line Strategies,” 2009.

[10] C. Seo and B. P. Zeigler, “Automating the DEVS modeling and simulation interface to web services,” in *Proceedings of the 2009 Spring Simulation Multiconference*, 2009, p. 158.

[11] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic press, 2000.

[12] T. Schwatinski and T. Pawletta, “An advanced simulation approach for parallel DEVS with ports,” in *Proceedings of the 2010 Spring Simulation Multiconference*, 2010, p. 147.

[13] “DEVS tools | DEVS Standardization Group.” [Online]. Available: http://cell-devs.sce.carleton.ca/devsgroup/?q=node/8. [Accessed: 30-Nov-2014].

[14] H. Cho and Y. Cho, “DEVS–C++ Reference Guide,” *URL Httpcactus Ece Ariz. Edu∼ Ykchodocdevsv*, vol. 41, 1997.

[15] G. Christen, A. Dobniewski, and G. Wainer, “Modeling state-based DEVS models in CD++,” in *proceedings of MGA, advanced simulation technologies conference*, 2004, pp. 105–110.

[16] G. A. Wainer, R. Madhoun, and K. Al-Zoubi, “Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services,” *Simul. Model. Pract. Theory*, vol. 16, no. 9, pp. 1266–1292, 2008.

[17] B. P. Zeigler, G. Ball, H. Cho, J. S. Lee, and H. Sarjoughian, “Implementation of the DEVS Formalism over the HLA/RTI: Problems and Solutions,” in *Simulation Interoperation Workshop (SIW)*, 1999, p. 065.
[18] B. P. Zeigler and H. S. Sarjoughian, “Support for hierarchical modular component-based model construction in DEVS/HLA,” in *Simulation Interoperability Workshop*, 1999, pp. 14–19.
[19] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, and others, “Modeling stateful resources with web services,” *Glob. Alliance*, 2004.
[20] S. Graham, G. Daniels, D. Davis, Y. Nakamura, S. Simeonov, P. Brittenham, P. Fremantle, D. Koenig, and C. Zentner, *Building Web services with Java: making sense of XML, SOAP, WSDL, and UDDI*. SAMS publishing, 2004.
[21] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, “Introduction to web services architecture,” *IBM Syst. J.*, vol. 41, no. 2, pp. 170–177, 2002.
[22] D. K. Barry, *Web Services, Service-oriented Architectures, and Cloud Computing: The Savvy Manager’s Guide*. Morgan Kaufmann, 2003.
[23] S. Potts and M. Kopack, *Sams teach yourself web services in 24 hours*. Sams, 2003.
[24] E. Cerami, *Web services essentials: distributed applications with XML-RPC, SOAP, UDDI & WSDL*. O’Reilly Media, Inc., 2002.
[25] B. Suda, “SOAP Web Services,” *Retrieved June*, vol. 29, p. 2010, 2003.
[26] H. Wang, J. Z. Huang, Y. Qu, and J. Xie, “Web services: problems and future directions,” *Web Semant. Sci. Serv. Agents World Wide Web*, vol. 1, no. 3, pp. 309–320, 2004.
[27] Q. Z. Sheng, B. Benatallah, R. Stephan, E. O.-Y. Mak, and Y. Q. Zhu, “Discovering e-services using UDDI in SELF-SERV,” in *Int. Conference on E-Business, Beijing, China*, 2002.