SIMULATION OnlineFirst, published on November 12, 2009 as doi:10.1177/0037549709354112

Web-based Distributed Network Analyzer using a System Entity Structure over a Service-oriented Architecture

Taekyu Kim
Center for Modeling and Simulation Studies,
Security Management Institute,
Dongheung Bldg 9F, Samsung-Dong 78-1, Gangnam-Gu, Seoul, 135-871,
Republic of Korea
taekyu@gmail.com

Chungman Seo
Bernard P. Zeigler
Arizona Center for Integrative Modeling and Simulation,
Electrical and Computer Engineering Department, The University of Arizona,
Tucson, AZ 85721,
USA
uracbul@gmail.com; *zeigler@ece.arizona.edu*

As a network's uses, and especially the number of internet users, increases rapidly, an efficient system for managing large-network traffic datasets becomes an important issue. Although there are several network traffic analysis tools, such as tcpdump, Ethereal, and other applications, these tools have weaknesses, namely the limited size of files, the use of command line execution, the large memory and huge computational power requirements. In addition to these scalability limitations, both tcpdump and Ethereal have security issues. Files captured by these tools keep all of the packet information, such as internet protocol (IP) addresses, port numbers, and packet sizes. As well as basic network traffic information, the captured files contain secure information: user identification numbers (IDs) and passwords. Therefore, the captured files should not be allowed to be leaked out. However, network analyses need to be performed outside the target networks in some cases. This paper presents an approach to efficiently and quickly analyze a large number of network behaviors. This is achieved by applying System Entity Structure (SES) theory. To speed up evaluation time, a web-based distributed simulation approach over Service-oriented Architecture (SOA) is applied. Discrete Event System Specification/Service-oriented Architecture (DEVS/SOA) is used to deploy workloads into multi-servers, increasing overall system performance. A web-based distributed simulation contains two fundamental processes: distributing and analyzing among loosely coupled models through message-passing methods. The distributed simulation – allocating distributing models inside networks and assigning analyzing models outside networks – also allows the analysis of network behaviors out of networks while keeping important information secured.

Keywords: distributed simulation, service-oriented architecture, discrete event system specification, system entity structure, intrusion detection system

SIMULATION, Vol. 00, Issue 0, Xxxxxxxx 2009 000–000

DOI: 10.1177/0037549709354112
Figures 2, 14, 25–28 appear in color online:
http://sim.sagepub.com

1. Introduction

As a network's uses, and especially the number of internet users, increases rapidly, an efficient system for managing large-network traffic datasets becomes an important issue. Although there are several network traffic analysis (NTA) tools, such as tcpdump, Ethereal, and other applications,

these tools are limited. Tcpdump is a powerful tool that allows us to sniff network packets and perform statistical analyses out of those dumps. One major drawback to tcpdump is the size of the flat file containing the text output. The other weakness is that tcpdump runs under the command line. Ethereal is a tool for network protocol analysis, software and protocol development, and educational purposes. Because it is an open source project, many network professionals around the world use Ethereal, and many researchers support it by adding enhancements. The functionality of Ethereal is very similar to the functionality of tcpdump, but it runs under a graphical user interface (GUI) front end. Ethereal has been supported by many network professionals, so it has many functions, such as protocol analysis, throughput analysis, and other statistical analyses. Ethereal is like a two-sided coin. It is very powerful, but also very complicated. Ethereal requires an initial learning curve but is a complete tool, and it is limited to running on local machines. In addition, Ethereal uses complete data for every analysis. Accessing a big data set requires memory overhead and is an inefficient use of computational power. Although Ethereal is easier to use than tcpdump, it still limits the size of the target-analyzing files. Our experiments show that Ethereal cannot analyze more than two days of network activities in personal computers. To examine more than two days of activities, network managers must control Ethereal by iterating capturing and analyzing processes periodically to avoid excessive system memory uses.

In addition to the scalable problem (the size limitation of the capture files), both tcpdump and Ethereal have security issues. Capture files, which are evaluated by either tcpdump or Ethereal, include all of the packet information, such as internet protocol (IP) addresses, protocol types, packet size, and other fundamental attributes. As well as basic network packet information, user identification numbers (IDs) and passwords are also contained in the captured files. Because the captured files hold secure information, tcpdump and Ethereal are allowed to monitor network behaviors and to capture raw network traffic inside networks with special privileges on some platforms. However, network analyses need to be performed outside target networks in some cases. This means that monitoring and capturing network behaviors are executed inside target networks, and evaluating network activities are completed out of the networks. To accomplish this distributed analysis, functionality should be deployed into multiple machines. At the same time, high priority information must be secured.

The main objective of this study is to propose an approach to deal with a large number of network behaviors being quickly and efficiently analyzed. The System Entity Structure (SES) facilitates implementing a system to achieve this goal. The SES is a theory for designing structured information hierarchically and efficiently. Specifically, the SES is very useful for data engineering. Firstly, we design a behavior that represents general network activities. The behavior design is based on the SES theory. Customers' requests are not always same. For example, some customers want to evaluate network protocol uses. On the other hand, some users want to measure network throughput. Depending on various requirements (pragmatic frames), systems need to be optimized for fast and effective analyses. The SES helps systems to be adaptively optimized. Accurate reactions to users' applications facilitate systems holding the right data only. Therefore, we could analyze long-term network activities, which Ethereal cannot evaluate. To speed up evaluation time, we apply a web-based distributed simulation methodology. A web-based distributed simulation contains two fundamental processes: distributing models into multi-servers and simulating among loosely coupled models through message-passing methods. Discrete Event System Specification/Service-oriented Architecture (DEVS/SOA) facilitates deploying workloads into multi-servers and consequently increasing overall system performance.

This paper includes theoretical background information in Section 2. The same section introduces Discrete Event System Specification (DEVS) formalism, the SES theory, and pragmatic frames for representing data engineering and web services. Section 3 states the problems found in previous studies. Section 4 illustrates design issues for implementing a distributed simulation for a NTA system, the distributed SES-based Network analyZER (SES/NZER), in detail. Section 5 presents the DEVS/SOA. We present the models built, simulating a distributed NTA system (protocols evaluation, network throughput measurement, and intrusion detection systems (IDSs)) based on DEVS formalism in Section 6. The experimental results are presented in Section 7. Lastly, we conclude this paper by addressing future research works.

2. Theoretical Background

This section presents the relevant theoretical background for web-based distributed simulation for network behavior analyses over a service-oriented architecture. Firstly, we present the DEVS, which is a mathematical formalism for modeling and simulation (M&S). The SES is introduced. The SES theorem is used for representing real-world states (network behaviors). Web service and Service-oriented Architecture (SOA) is provided, respectively.

2.1 Discrete Event System Specification

The DEVS is a formalism providing a means of specifying a mathematical object called a system [1]. It also allows the building of modular and hierarchical model compositions based on the closure-under-coupling paradigm. The DEVS modeling approach captures a system's structure from both the functional and physical points of

view. A system is described as a set of input/output events and internal states along with behavior functions regarding event consumption/production and internal state transitions. Generally, models are considered as either atomic or coupled. The atomic model can be illustrated as a black box having a set of inputs (X) and a set of outputs (Y). The Atomic model includes a description of the interface, as well as the data flow between itself and other DEVS models. The atomic model also specifies a set of internal states (S) with some operation functions (i.e., the external transition function (δ_{ext}), the internal transition function (δ_{int}), the output function (λ), and the time advance function ($ta(\cdot)$)) to describe the dynamic behavior of the model.

The external transition function (δ_{ext}) carries the input and changes the system states. The internal transition function (δ_{int}) changes the internal variables from the previous state to the next when no events have occurred since the last transition. The output function (λ) generates an output event to outside models in the current state. The time advance ($ta(\cdot)$) function adjusts the simulation time after generating an output event. The atomic model is specified as follows:

$$M = \langle X, S, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, ta \rangle$$

where X is the set of external input events, S is the set of sequential states, Y is the set of outputs, $\delta_{\text{int}} : S \rightarrow S$: is the internal transition function, and $\delta_{\text{ext}} : Q \times X^b \rightarrow S$: is the external transition function, where $Q = \{(s, e) | \ s \in \text{S}, 0 \leq e \leq ta(s)\}$; is the set of total states, e is the elapsed time since last state transition, X^b is a set of bags over elements in X, $\lambda : S \rightarrow Y$: is the output function generating external events at the output, and $ta : S \rightarrow R^+_{0,\infty}$: is the time advance function.

Basic models may be joined in the DEVS formalism to form a coupled model. A coupled model is the major class that embodies the hierarchical model composition constructs of the DEVS formalism [1]. A coupled model is made up of component models, and the coupling relations that establish the desired communication links. A coupled model illustrates how to couple (connect) several component models together to form a new model. Two significant activities involved in coupled models are specifying its component models and defining the couplings that create the desired communication networks. A coupled model is defined as follows:

$$DN = \left\langle X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} \right\rangle$$

where X is a set of external input events, Y is a set of outputs, and D is a set of components names; for each i in D, M_i is a component model and I_i is the set of influences for I; for each j in I_i, $Z_{i,j}$ is the i-to-j output translation function.

A coupled model template contains the following information [2]:

- the set of components;
- the set of input ports through which external events are received;
- the set of output ports through which external events are sent.

The coupling specification consisting of:

- the external input coupling (EIC) connects the input ports of the coupled model to one or more of the input ports of the components;
- the external output coupling (EOC) connects the output ports of the components to one or more of the output ports of the coupled model;
- the internal coupling (IC) connects the output ports of the components to the input ports of the other components.

2.2 System Entity Structure

The basic concept of the SES is that a system entity represents the real system enclosed within a certain choice of system boundary. Many system entities and experimental frames are dealt with in a real system. Thus, it is necessary to organize the model and experimental frames around the structure. The entity structure is a template from which the decomposition trees of the existing models can be extracted. Moreover, the entity structure is a template for constructing models from those already existing. The key components that the SES consists of are as follows:

Entity: An entity is intended to represent a real world object that either can be independently identified or is postulated as a component in some decomposition or a real world object.

Aspect: An aspect represents one decomposition of an entity. The children of an aspect are entities representing components in a decomposition of its parents.

Specialization: A specialization is a mode of classifying entities and is used to express alternative choices for components in the system being modeled. The children of a specialization are entities representing variants of their parent.

To construct a desired simulation model to meet the design objective, the pruning operation is used to reduce the SES to a pruned entity structure (PES) [3]. The PES can be transformed into a composition tree and eventually synthesized into a simulation model. Professor Zeigler proposed the SES [3, 4], which is a theory to design systems hierarchically and structurally. The SES includes entities and their relationships.

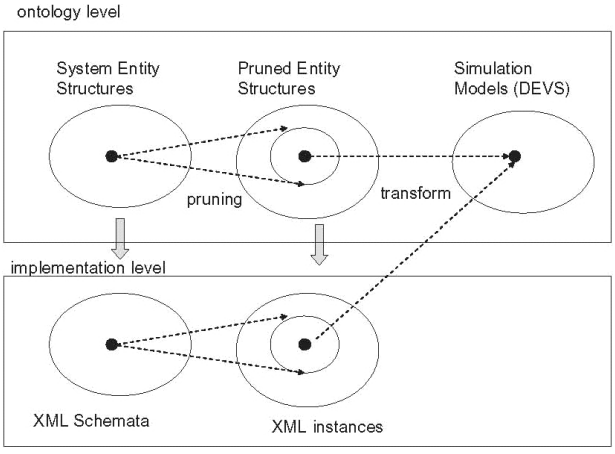

Figure 1. Architecture for model and simulation-based data engineering methodology [6]

Figure 1 illustrates the SES basic methodology of the conceptual relationship between the SES representing ontologies and the implementation in the eXtensible Markup Language (XML) [5]. First of all, the SES, which can describe the components in the source data, is developed. The SES structure produces important information to build the Document Type Definition (DTD) or schema. Entity, aspect, multi-aspect, and specialization build the primary components in the DTD or schema. At the ontology level, the modeler develops one or more SESs depending on the models, and the SESs are merged to create an ontology in order to satisfy the pragmatic frames of interest in a given application domain. A SES can be specified in various ways, and then it is transformed to an XML schema or an XML DTD or XML Schema Definition (XSD) at an implementation level. The pruning operation of SESs creates PESs, and the PESs transform to simulation models.

2.3 Web Services

A web service [7] is a software system for communicating between a client and a server over a network with XML messages called Simple Object Access Protocol (SOAP) [5, 8]. The web service makes the request for machine-to-machine or application-to-application communication possible with neutral message passing, even though each machine or application is not in the same domain. Such interoperability among heterogeneous applications is realized by the web service providing a standard means of communication and platform independency.

Web services technologies architecture [9] is based on exchanging messages, describing web services, and publishing and discovering web service descriptions. The messages are exchanged by SOAP messages conveyed by IPs. Web services are described by Web Services Description Language (WSDL) [10], which is an XML-based language providing the required information, such as message types, signatures of services, and the location of services, for clients to consume the services. Publishing and discovering web service descriptions are managed by Universal Description Discover and Integration (UDDI) [11], which is a platform-independent and XML style registry. In other words, three roles are classified in the architecture: that is, a service provider, a service discovery agency (UDDI), and a service requestor. The interaction of the roles involves publishing, finding, and binding operations. A service provider defines a service description for a web service and publishes it to a service discovery agency. This operation is a publishing operation between the service provider and the service discovery agency. A service requestor uses a finding operation to retrieve a service description locally or from a discovery agency and uses the service description to bind it with a service provider and invoke or interact with the web service implementation. Figure 2 illustrates the basic web services architecture describing the three roles and operations with WSDL and SOAP.

Whereas a web service is an interface described by a service description, its implementation is the software module provided by the service provider (server) in a network-accessible environment. It is invoked by or interacts with a service requestor (client).

Web services are invoked in many ways, but the most common use of web services is categorized into three methods, namely the Remote Procedure Call (RPC), the SOA [12], and the Representational State Transfer (REST) [13]. RPC web services was the first web services approach that had a distributed function call interface described in the WSDL operation. Although it is widely used and upheld, it does not support a loosely coupled concept due to the services being mapped directly to language-specific functions calls. A web service is an implementation of SOA concepts, which means a message is an important unit of communication regarded a 'message-oriented' service. This approach supports a loose coupling concept focusing on the contents of the WSDL. Web service focuses on the existence of resources rather than messages or operations. Web service considers the WSDL as a description of SOAP messaging over the Hypertext Transfer Protocol (HTTP). The WSDL is implemented as an abstraction on top of the SOAP. REST is a style of software architecture for distributed hypermedia systems, such as the world wide web [14]. As such, it is not strictly a method for building web services. REST is an approach for getting information content from a web site by reading a designated web page that contains an XML file that describes and includes the desired content. REST is simpler to use than the well-known SOAP approach, which requires writing or using a provided server program and a client program.

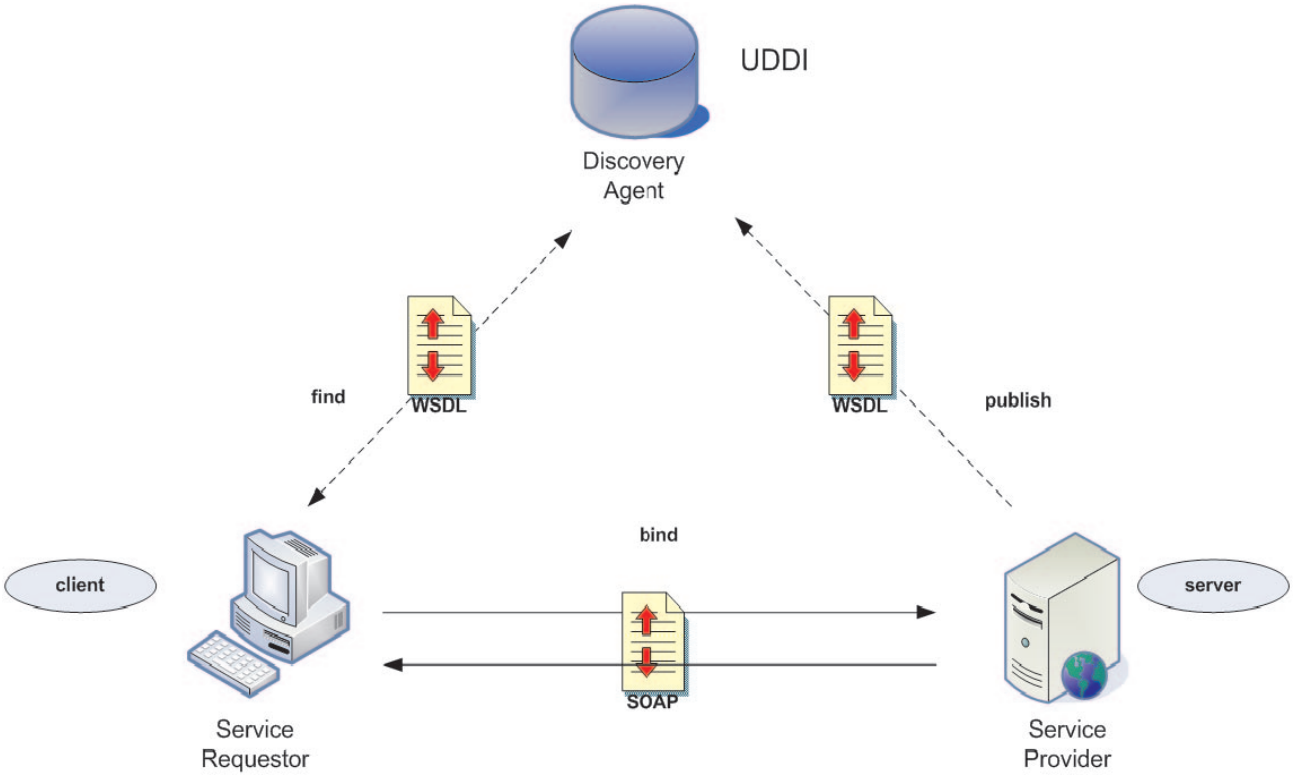

Figure 2. Web services architecture

Because the SOA environment provides languages and platforms in a neutral system, the DEVS/SOA is used to solve interoperability problems in the System of Systems (SOS) [15], as well as to provide a distributed computing environment [16]. Seo and Zeigler [17] developed an interoperable DEVS/SOA system to simulate DEVS models in different languages (e.g. DEVSJAVA [18] and aDEVS [19]). The interoperable DEVS/SOA requires DEVS simulator services, neutral message passing between the DEVS simulator services, and a DEVS namespace [20]. It provides the multi-layered interoperability introduced in Tolk et al. [21].

There are some researches using a SOA environment and Common Object Request Broker Architecture (CORBA) to implement distributed and parallel simulation. Wutzler and Sarjoughian [22] introduced the *Shared Abstract Model* (SAM) to simulate DEVS models in different languages. The SAM emphasizes model interoperability through the abstract model realized by CORBA to provide communication channels between abstract models. This approach has limitations of partitioning and allocating a complex model on different machines. Yoo et al. [23] used web services to implement Optimization via Simulation (OvS) through Parallel Replicated Discrete Event Simulation (PRDES). The web service can contain an optimization module or a simulation module. The web services with simulation modules are like business processes. They receive input data from the optimization module and send simulation results to a repository in the supporting server. Wainer et al. [24, 25] proposed a distributed simulation engine named DCD++ using the DEVS, Cell-DEVS formalisms, and web service technologies. The distributed simulation engine utilizes web services to pass the models, simulation protocols, and simulation messages. It employs the Java Native Interface to allow the web service in a Java program to call and to be called by native applications and libraries in other languages (CD++). Each web service has a master coordinator or a slave coordinator, and a simulator to reduce message overheads.

2.4 Intrusion Detection System

In this section, we discuss an advanced concept, namely intrusion detection evaluation. Widespread use of networked computers has made computer security a serious issue. Every networked computer, to varying degrees, is vulnerable to malicious computer attacks that can result in a range of security violations, such as unauthorized user access to a system or the disruption of system services. Traditionally, computer security approaches have focused on preventing such attacks from occurring through the use

of firewalls and security policies. However, for most systems, complete attack prevention is not realistically attainable due to system complexity, configuration and administration errors, and abuse by authorized users. For this reason, attack detection has been an important aspect of recent computer security efforts [26, 27].

IDSs are designed to detect computer attacks. They monitor the activities of computers and networks for attacks that are inevitable, despite security precautions. If attacks are discovered, IDSs can alert administrators, defend against the attacks, or provide information that may help prevent future attacks. IDSs are not all equal in capability or reliability. A particular system may only detect a specific subset of possible attacks. In addition, it may have a different level of detection accuracy or a different false alarm rate than other systems. Results from IDS evaluations allow users to make informed decisions on what system to use and are extremely important for guiding research. IDSs have become an essential component of computer security to detect these attacks before they inflict widespread damage. A review of current approaches to intrusion detection is available in Bishop et al.'s article [28]. Some approaches detect attacks in real time and can stop an attack in progress. Others provide after-the-fact information about attacks and can help to repair damage, understand the attack mechanism, and reduce the possibility of future attacks of the same type. More advanced IDSs detect never-before-seen, new, attacks, while the more typical systems detect previously seen, known attacks.

While advances in network IDS development have led to more stable network security, fast and effective analysis methods are needed to save maintenance budgets and recover from problems caused by attacks and anomalous behavior errors. These critical issues are yet to be addressed due to the lack of appropriate frameworks. Indeed, IDS researchers have difficulty in testing their algorithms before applying them to real systems. In IDS testing, the main problems are:

- Problem 1. Limitation of data storage:
  - there are multitudes of events in networks and hosts;
  - each event includes many attributes of packet information.
- Problem 2. Lack of analysis methods:
  - difficulty in generating attacks;
  - difficulty in implementing complete IDSs.
- Problem 3. Excessive resource consumption:
  - existing systems require huge computational resources in time (central processing unit (CPU)) and space (memory).

A data engineering-based M&S framework is intended to support the testing and evaluation of network IDSs. Data engineering, supported by network ontology modeling, enables our approach to be efficient in managing and processing huge amounts of network traffic data. As an example, the Knowledge Discovery and Data Mining (KDD)'99 dataset was generated by the Lincoln Laboratory at the Massachusetts Institute of Technology (MIT) for the purpose of testing network IDSs. The dataset includes various attacks packet events, as well as normal transmissions. From this dataset, network traffic generators are produced automatically in response to customers' (IDS developers and testers) requirements. Different customers may need different attributes for their particular IDSs (pragmatic frames). Including unnecessary data in packet information consumes computational power and memory. This is the reason why we employ a data engineering-based simulation framework for IDSs. Our goal is to support a simulation framework for testing and evaluating network IDSs. Ontology/data engineering methodology empowers our design to be efficient for managing and using large-size data.

3. Problem Statements

The goals of a NTA are to help network administrators to manage very complicated network topology and to increase efficiency for secure and effective data transfer. Network use, especially the number of internet users, is increasing rapidly. In addition, a high quality of service is required, and large-packet data need to be exchanged among servers and clients to meet recent needs, in particular the high quality of services. As such, this high quality requirement results in sudden network traffic increases. As a result, designing efficient systems for managing large-network traffic data becomes an important issue. The ontology/data engineering methodology is used to build an effective system for analyzing large amounts of network traffic data. The SES/NZER is used to develop a system that allows easy and efficient information sharing among organizations. The SES and XML modeling approaches allow systems to easily handle a huge amount of data, and the two approaches facilitate the M&S study, because the architecture of the SES is a hierarchical tree structure. In addition, the characteristics of XML, such as scalability and portability, are very good for managing metadata. We compare execution times and measure system memory (random-access memory (RAM)) usages between Ethereal and the SES/NZER. We use a half day, one day, and two days of data to evaluate system performance variations. Table 1 shows the measurements of memory use and execution times for network protocol analyses. Table 2 illustrates the experimental results for the throughput evaluations.

The loading time of Ethereal refers to the time taken to invoke the captured data file. The loading time of the

Table 1. Memory usages and execution times for protocol analysis

	Ethereal			SES/NZER		
	Half day	One day	Two days	Half day	One day	Two days
Loading time	1 min 18 s	2 min 28 s	N/A	5 min 28 s	10 min 44 s	20min 59sec
Num of events	1,063,803	2,045,700	N/A	1,063,803	2,045,700	4,091,400
Memory usage	706 MB	1323 MB	N/A	98 MB	98 MB	98MB
Analyzing time	25 s	50 s	N/A	5 min 29 s	10 min 58 s	22min 59sec

Table 2. Memory usages and execution times for throughput analysis

	Ethereal			SES/NZER		
	Half day	One day	Two days	Half day	One day	Two days
Loading time	1 min 18 s	2 min 28 s	N/A	5 min 32 s	11 min 27 s	22min 14min
Num of events	1,063,803	2,045,700	N/A	1,063,803	2,045,700	4,091,400
Memory usage	706 MB	1323 MB	N/A	104 MB	104 MB	104MB
Analyzing time	19 s	55 s	N/A	5 min 17 s	9 min 56 s	22min 13min

SES/NZER is the time spent generating PES XML document files regarding users' requests. The SES/NZER takes a greater amount of time than Ethereal to load the data to evaluate. In addition, Ethereal is faster to analyze data than the SES/NZER. We noticed that both loading time and analyzing time increase linearly corresponding to the total number of events during the capturing period. Tables 2 and 3 indicate that Ethereal is faster than the SES/NZER. However, Ethereal is a complete tool, so it should be run on a single machine only. On the other hand, the SES/NZER is scalable to distributed environments. A web-based distributed SES/NZER may reduce both loading data time and analyzing time by deploying workloads. Ideally, run time decreases as an inverse ratio of the number of servers. Ultimately, the SES/NZER can be faster than Ethereal under distributed environments. The important things we must see are the values of memory use measurements. For half-day data, Ethereal requires 706 MB of system memory (RAM). As data size increases, the memory requirement of Ethereal increases linearly. However, the SES/NZER needs 98 MB of a system memory for half-day data, and the memory requirement of the SES/NZER never increases in correspondence to source data sizes. The SES/NZER keeps the system stable. For two-day captured data, Ethereal cannot load the data and consequently cannot analyze the network activities. Ethereal is shut down due to memory overflow problems. On the other hand, the SES/NZER can evaluate network behaviors, although it takes time. Figure 3 illustrates the structural comparison between Ethereal and the SES/NZER for one-day data analyses. Tables 1 and 2 and Figure 3 present the reason why developing a web-based distributed SES/NZER is a promising research area of network analysis fields for increasing computational power. Deploying workloads naturally tends to make efficient use of memory caches,

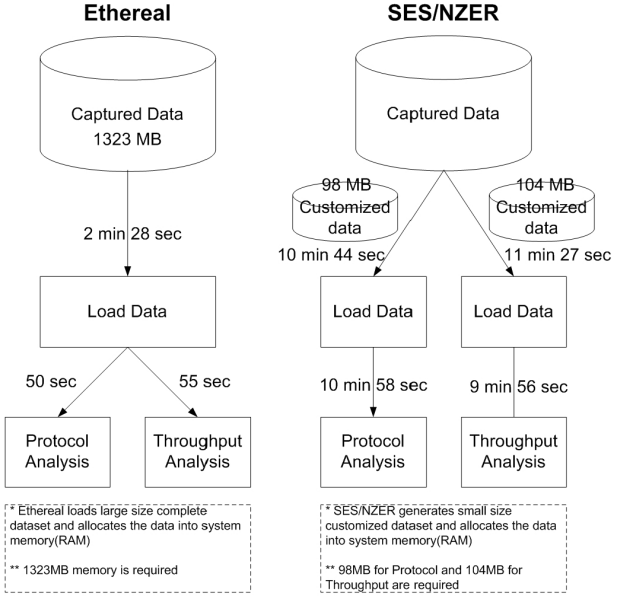

Figure 3. Structural comparison between Ethereal and the SES/NZER

as well as speeds up both data loading time and analysis time.

The fact that the SES/NZER is more efficient in system memory requirements than Ethereal facilitates the SES/NZER in analyzing a large amount of data. However, the SES/NZER is weak in evaluation speed. One solution to achieve feasible speed-up and efficiency is parallel processing. Parallel processing consists of dividing data into two or more smaller datasets, assigning datasets into multiple processors, and processing multiple datasets in

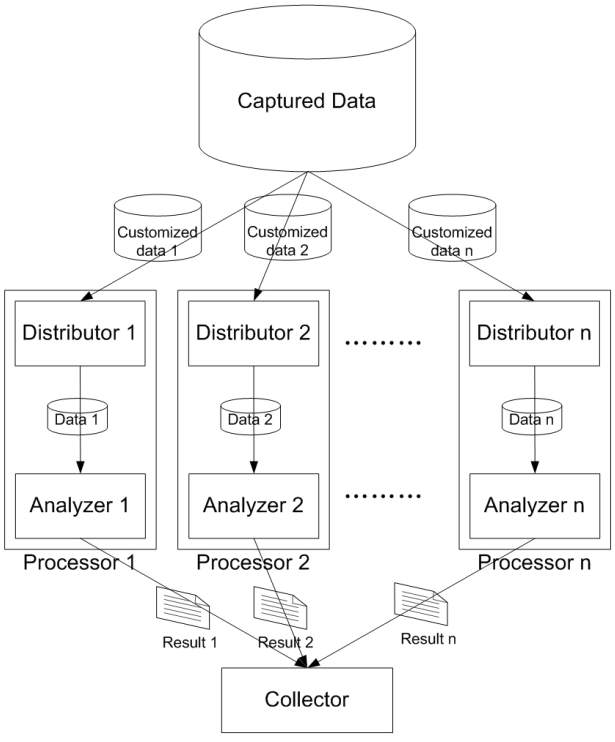

Figure 4. Divide and conquer SES/NZER

multiple processors simultaneously. Divide and conquer (D&C) is an important algorithm design paradigm. D&C was first introduced by Karatsuba [29] as an algorithm for multiplying two n-digit numbers with an algorithmic complexity $O(n)$ on $n^{\log_2 3}$. The D&C scheme is also widely used in parallel processing designs for reducing the complexity of processors. D&C solves a problem easily by dividing a problem into two or more smaller problems. Each of these smaller problems is solved, and the solutions for the smaller problems are combined to produce a solution for the original problem. Figure 4 shows a D&C scheme for the SES/NZER.

The first step is the dividing process. A large amount of source data are segmented by n numbers of small datasets. Fragmented individual datasets are assigned to n numbers of processors. Each processor analyzes its corresponding dataset. The workload of each processor may be reduced as an inverse ratio of the number of processors. Subsequently, all of the analyzed results of the processors are integrated together. This integrating of all of the results and concluding with a final output is the conquering process. This D&C approach requires not only segmentation overheads for dividing data, but also communication overheads for conquering all of the results. Even though there are overhead disadvantages, this method includes two strengths that overcome the disadvantages. One advantage is that this approach enables applications, which need to process a large amount of data and require high computational power in time (CPU) and in space (memory), to be run on inexpensive personal computers rather than on high-cost server machines. The other advantage is quick evaluation time. Multiple processors execute their work simultaneously. Therefore, parallel processing methods reduce processing time compared to sequential processing methods. In addition, the D&C approach may be applied to distributed environments. Processors are deployed into multiple machines that are connected by loosely coupled links. Loosely coupled systems are harder to implement than tightly coupled systems, because systems should be synchronized for validation issues. However, once it is implemented, each processor is independent to other processors, and none of the processor's activities affect other processors' behaviors. In this paper, we use web service schemes over SOA to construct distrusted environments. This web-based distributed simulation increases independency and decreases complexity in each host. Table 3 illustrates comparisons between the SES/NZER and a web-based distributed SES/NZER.

Table 3. SES/NZER versus distributed SES/NZER

	SES/NZER	Distributed SES/NZER
Locality	Local host	Distributed hosts
Parallelism	None	High
Process time	Slow	Fastest
Overheads	No additional overhead	Data segment overheads Communication overheads

4. Design Issues

In this paper, we show two kinds of network behavior analyses: generic network behavior analyses and specialized analyses. For generic purpose network behavior evaluation, a protocol analysis and throughput analysis are examined. In addition, IDSs are evaluated for specialized cases. Figure 5 represents the hierarchical system structure. A web-based distributed SES/NZER fulfills either analyzing generic network traffic activities (protocol analysis or throughput analysis) or evaluating an IDS.

Simplifying the complexity of models is necessary in order to meet the required level of simulation performance, since complexity constrains modeling to be severely limited [1]. The complexity of a model can be measured by the resources required by a particular simulator to correctly interpret it. That is, complexity is measured relative to a particular simulator, or class of simulators. Even though computers continue to become faster and increase in memory, they are still not good enough to make our models into reality. Successful modeling can be seen as valid simplification. Simplifying or reducing the complexity enables models to be executed in our limited

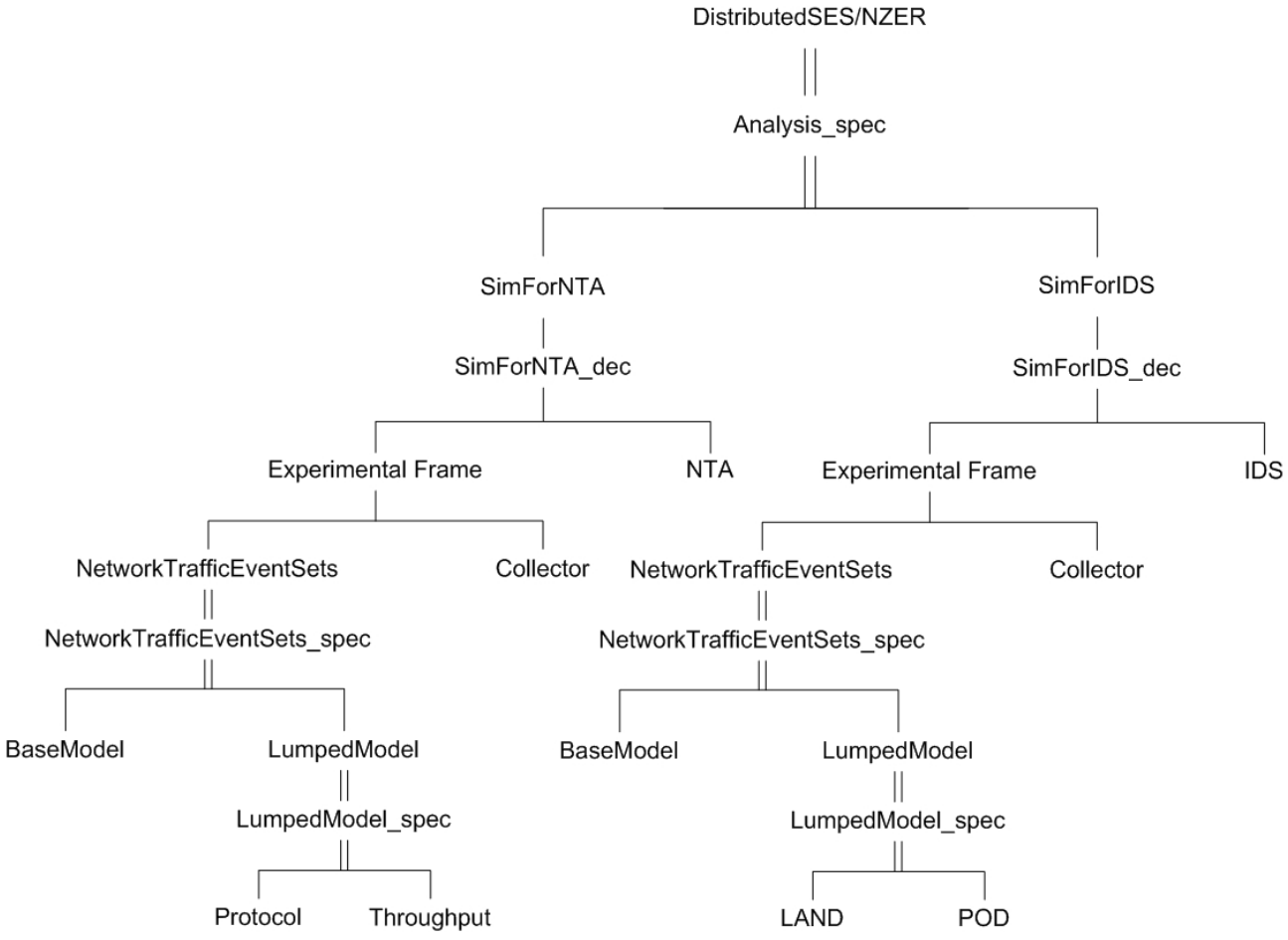

Figure 5. Distributed SES/NZER system hierarchy

resource (time and size) simulation environments. However, simplified models must be valid within some experimental frame of interest. An experimental frame represents a specification of the conditions under which the system is observed or experimented with. As such, an experimental frame is the operational formulation of the objectives that motivate a M&S project. Figure 6 shows the pair of models involved. They are base and lumped models in an experimental frame.

The base model requires more resources in time and size for interpretation than the lumped model. Moreover, the base model is more valid within a larger set of experimental frames (with respect to a real system) than the lumped model. As such, the lumped model might be just as valid as the base model within a particular frame of interest (a particular pragmatic frame). The concept of morphism, a relation that places elements of system description into correspondences, provides criteria for judging the equivalence of base and lumped models with respect to an experimental frame. Base models include many elements, but all of the elements in a base model are not always required in pragmatic frames. Mapping a methodology from a base model to lumped models reduces the number of elements included so that it increases computational power in time and size.

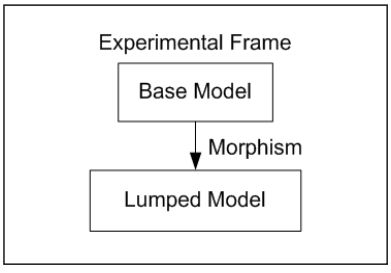

Figure 6. Base/lumped model equivalence in experimental frame

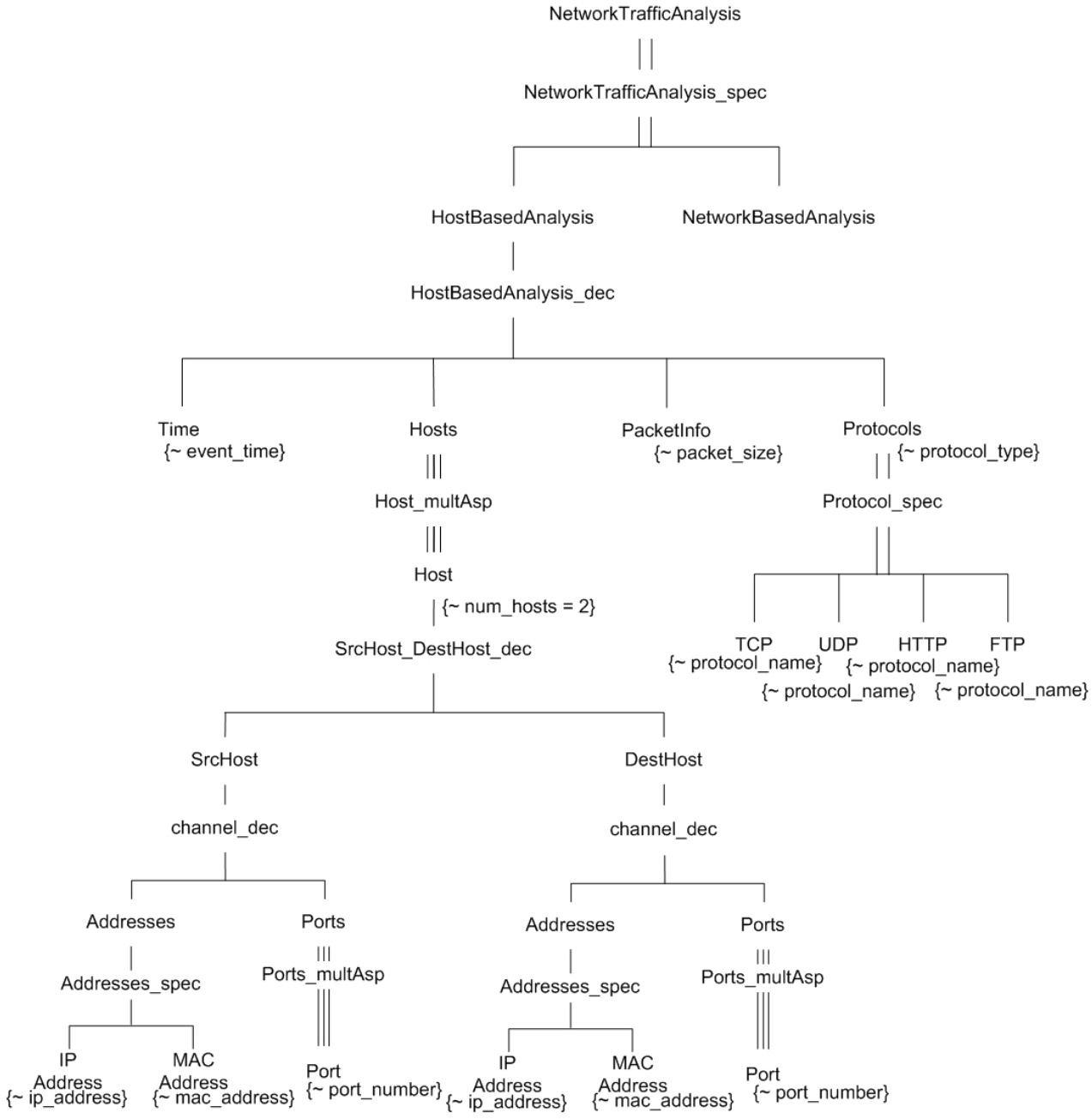

Figure 7. SES for network traffic behavior

4.1 Network Behavior Design (Base Model)

In this section, we design network behaviors using SES theory. The SES represents network traffic behaviors for the purpose of a host-based analysis. Nine elements, which are an event time, a source IP address, a source Media Access Control (MAC) address, a source port number, a destination IP address, a destination MAC address, a destination port number, a protocol, and packet length, are examined in a NTA. These nine essential elements are included in network packet headers. Categorizing these nine elements is important for fast and accurate network behavior evaluation and analysis. We use the SES methodology to classify network packet information in the hierarchical tree structure. Figure 7 is a hierarchical SES tree structure representing network packet behaviors.

The root entity *NetworkTrafficAnalysis* is the top-level entity that analyzes network traffic, and using the *NetworkTrafficAnalysis_spec*, the *NetworkTrafficAnalysis* can be implemented with the *HostBaseAnalysis* or the *NetworkBasedAnalysis*. Since the aim of this example is to analyze network traffic on hosts, we do not branch the *NetworkBasedAnalysis* any further. The *HostBasedAnalysis* is composed of four entities: the *Hosts*, the *Time*, the *Protocols*, and the *PacketInfo*. The *Hosts* is composed of multi-Host, and the *Host* has an attribute identifying the number of hosts, and that value is set as two because the *Host* is always composed of the *SrcHost* and the *DestHost*. The *SrcHost* is composed of two entities, such as the *Addresses* and the *Ports*. The *Addresses* can be specialized as the *IPAddress* or the *MACAddress* using the *Addresses_spec*. Both the *IPAddress* and the *MACAddress* have their own attribute of the *ip_address* and the *mac_address*. The *Ports* is composed of multi-Port, and the *Port* has an attribute, the *port_number*. The *DestHost* has the same tree structure as the *SrcHost*. One of the *HostBasedAnalysis*'s children is the *Time*, and the *Time* has an attribute of the *event_time*. Another child entity of the *HostBasedAnalysis* is the *Protocols*, and the *Protocols* have the *protocol_type* attribute. The *Protocols* can be implemented with the Transmission Control Protocol (*TCP*), the User Datagram Protocol (*UDP*), the *HTTP*, or the File Transfer Protocol (*FTP*) using the *Protocol_spec*. Those four entities have their own attribute, the *protocol_name*. We filter and capture network traffic data related to four very common protocols. The last entity of the *HostBasedAnalysis*'s children is the *PacketInfo*. In this study, we aim to analyze throughputs so that the *packet_size* is the only attribute of the *PacketInfo* entity.

This SES represents based models of both simulation for network traffic analysis (*SimForNTA*) and simulation for the IDSs in Figure 5. For the use of generic NTA simulation, we monitor network activities and capture the fundamental packet information in a subnet of the Arizona Center for Integrative Modeling and Simulation (ACIMS) laboratory [30] in the department of electrical and computer engineering at the University of Arizona. We use the Ethereal [31], which is a well-known network protocol analyzer, for capturing network behaviors. Unlike generic network behavior analyses, source data for IDS simulation must include attack packet transmissions, as well as normal packet transmissions. However, generating attack packets is strictly prohibited even if it is for academic research purposes. Therefore, for the purpose of IDS simulation, we use a KDD'99 dataset [32]. The MIT Lincoln Laboratory supported by the Defense Advanced Research Projects Agency (DARPA) project [33] simulated and generated a network traffic dataset, including attacks, in 1998. This dataset has been widely used in the area of computer network IDS research and is now regarded as the standard. In addition, it is well known by the name KDD'99 dataset, because KDD [34] processed the network traffic data generated by the MIT's Lincoln Laboratory and opened a contest. Many network researchers and artificial intelligent researchers use this dataset for their IDSs. The dataset includes two weeks (five days/week) simulation data. Every day's data set is huge, e.g., the first week's Monday data has 60,000 events. According to the SES in Figure 7, the KDD'99 dataset is re-structured by extracting required data, which map to the entities of Figure 7, from the full KDD'99 dataset.

4.2 Pragmatic Frames (Lumped Models)

Target network behavior analyses are defined by customers. Every analysis should have a different set of information with regards to users' requests. These different requests are pragmatic frames. Keeping unnecessary information decreases computational performance. For speed and effectiveness, customers' requirements need to create corresponding SESs that keep accurate entities and attributes. Consequently, users' target analyses must be modeled and simulated based on the new SES and their XML document instances (PESs). Newly created SESs based on customers' requirements (pragmatic frames) represent lumped models in a modeling point of view. The unified processes, creating new SESs, and setting up simulation environments dynamically by assigning a lumped model instead of a base model, which is shown in Figure 6, increase efficiency and automated factors.

4.2.1 Generic Network Behavior Analyses

We illustrate two cases of generic network behavior analyses: protocols analysis and network throughput measurement. The first analysis, evaluating the number of packets per protocols, requires two attributes of protocol names and IDs. The second analysis, measuring network throughput, needs event times and packet sizes.

Once the customers or users request a protocol usage analysis, a new SES is created automatically as given in Figure 8. The SES, *ProtocolAnalyses*, has a multi-aspect of *ProtocolAnalysis*. The entity, *ProtocolAnalysis*, is composed of two entities, *ID* and *Protocol. ID* has an attribute, *id_number*, and *Protocol* has an attribute, *protocol_type.*

Figure 9 shows a SES for the network throughput evaluation. The SES name is *ThroughputAnalyses*. *ThroughputAnalyses* has a multi-aspect of *ThroughputAnalysis*. *ThroughputAnalysis* is composed of *EventTime* and *PacketSize*. *EventTime* has an attribute, *event_time*, and *PacketSize* has an attribute, *packet_size*.

4.2.2 Intrusion Detection Systems

This study examines two intrusion detecting agents for a LAND attack and a Ping of Death (POD) attack. The LAND attack is a Denial of Service (DoS) attack that consists of sending a special poison spoofed packet to a

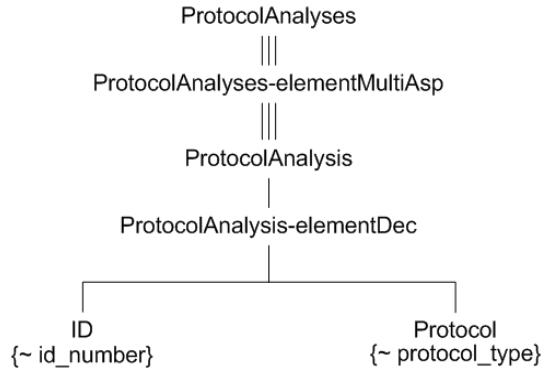

Figure 8. SES for protocol analyses

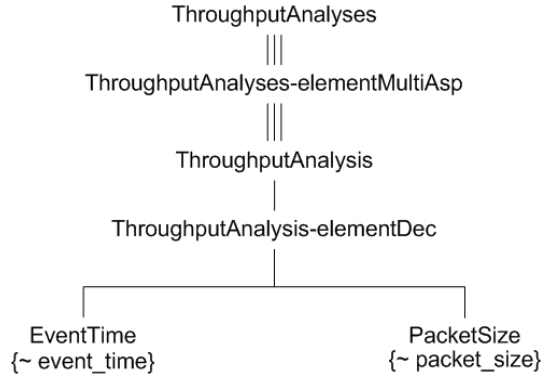

Figure 9. SES for throughput analyses

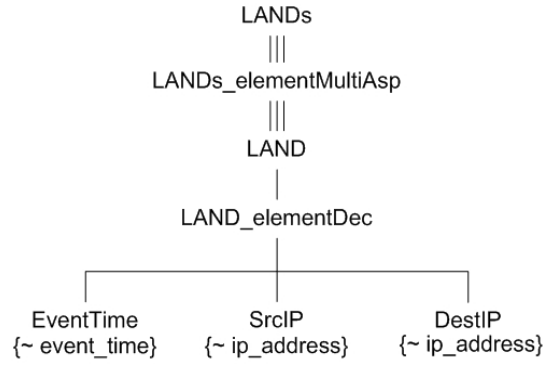

Figure 10. SES for LAND attack detection

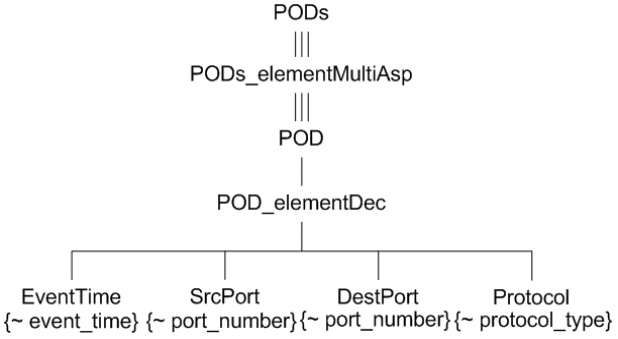

Figure 11. SES for POD attack detection

computer, causing it to lock up. The LAND attack occurs when an attacker sends a spoofed synchronous (SYN) packet in which the source address is the same as the destination address [35]. This is a rather old attack, and current patches should stop them for most systems. Symptoms of the LAND attack are different by operating systems. The LAND takes affect by slowing down operating speed, crashing and shutting down systems, or denying users access to services on machines. The LAND attack is recognizable because IP packets with an identical source IP address and a destination IP address must never exist on a properly working network. Therefore, we need two attributes, a source IP address and a destination IP address, to detect LAND attacks. In addition to the source IP address and the destination IP address, an attribute, event time, is needed for diagnosis purposes. Figure 10 illustrates a SES for LAND attack detection.

The POD attack is a type of DoS attack in which the attacker sends a ping request that is larger than 65,536 bytes, which is the maximum size that IP allows. While a ping larger than 65,536 bytes is too large to fit in one packet that can be transmitted, TCP/IP allows a packet to be fragmented, essentially splitting the packet into smaller segments that are eventually reassembled. The POD attack is relatively easy to carry out and very dangerous due to its high probability of success. Operating system vendors have made patches available to avoid the POD, but many web sites continue to block Internet Control Message Protocol (ICMP) ping messages at their firewalls to avoid similar DoS attacks. An attempted POD can be identified by noting the size of all ICMP packets and flagging those that are larger than 64,000 bytes [35]. However, the KDD'99 dataset does not have the attribute of packet size. The ICMP does not have a port abstraction. The ICMP (ping, trace) is a layer 3 protocol suite within the TCP/IP suite, and ICMP does not test any layer 4 or above functions; therefore, it has no TCP/UDP layer 4 port number. So, we may detect POD attacks with three attributes: a source host port number, a destination host port number, and a protocol. Figure 11 presents an SES for POD attack detection.

4.2.3 Mapping

Once a new SES is generated to correspond to a customer's requirements, the next step is producing new PESs

based on the new SES. Firstly, we need to extract correct data values from large PES instances (XML documents) of a source SES. Then, newly customized PESs are generated with the extracted attribute values from the source PESs. However, the problem is the case in which the structures of two SESs, a source SES and a target SES, are different. In this case, it is constrained from generating the new PESs by transforming directly from the source PESs. As a result, we must apply an alternative operation. Mapping enables the retrieval of required data values from the source PESs and assigns the correct values to the target PESs.

Mappings could be in two kinds of forms: transformations and restructurings. Transformations are mappings from one representation to another and are referred to as general mappings. Restructurings are mappings whose domain and range are the same. This means that a restructuring changes the structure of an object without changing the form in which it is expressed. A concept of equivalence must support such restructurings, i.e., the before and after structures must be equivalent with respect to some aspect of interest to the modeler. Such restructurings apply to reducing the size of a tree, which enables optimization for finding the best representation of some given information within a representation domain. This general restructuring process eliminates labels, including those of aspect, multi-aspect, and specialization. Eliminating such labels in a schema for a SES reduces the amount of overhead in carrying payload information. The resulting SES is equivalent to the original in the sense that the same family of PESs is defined. However, this mapping has a limitation in that it is 'not reversible', because such restructuring removes information that may be needed in downstream processing of the transmitted data.

We design the SES, *NetworkTrafficAnalysis*, for the generic purposes of network traffic behavior analyses as described in Figure 7. New SESs are generated to correspond to customers' requirements. However, the problem is that the structures of the two SESs, the *NetworkTrafficAnalysis* and one of the *ProtocolAnalyses*, the *ThroughputAnalyses*, the *LANDs*, or the *PODs*, have different structures. Performing mapping operations results in PES outputs, and the outputs are instances in XML document format. Then, the PES XML instance files are used as a role of input source data for M&S purposes.

5. Discrete Event System Specification Service-oriented Architecture

5.1 Virtual Time DEVS Simulation on SOA

DEVS simulation on a SOA [36, 37] consists of three layers, namely model distribution, simulation, and simulation result return. To support these layers, two services, the *MainSerivce* and *Simulation*, are implemented. *MainService* has four services: the *Upload* DEVS model, the *Compile* DEVS model, the *Simulate* DEVS model, and the *Get*

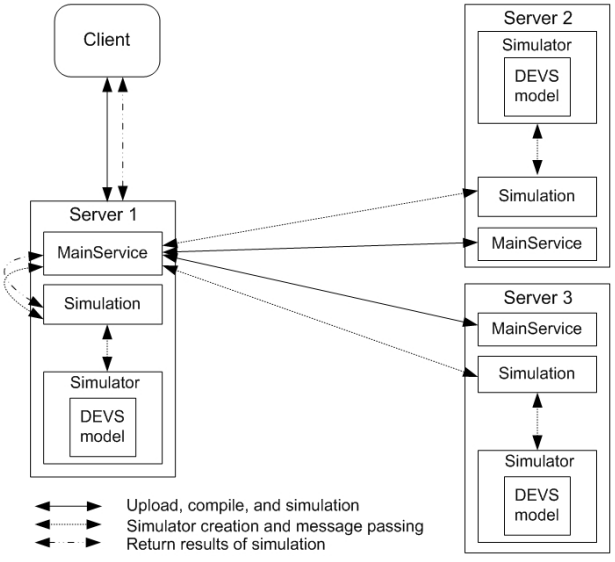

Figure 12. Overall architecture of DEVS simulation on SOA

result of the simulation. The *Simulation* service covers the DEVS simulation protocols. It has nine services: *Initialize simulator*, *Run transition* in simulator, *Run lambda function* in simulator, *Inject message* to simulator, *Get time of next event* from simulator, *Get time advance* from simulator, *Get console log* from all the simulators, *Finalize simulation* service, and *Get result* of simulation.

Figure 12 represents the overall sketch of DEVS simulation on SOA. As seen in Figure 12, this system has two components: a client and some servers. Each server has two services (*MainService* and *Simulation*) and the DEVS M&S environment. The beginning of DEVS simulation on SOA is to upload DEVS models to each server. A client assigns each model to an available server that has two services for the DEVS simulation. A main server assigned to a top DEVS model becomes a coordinator during the DEVS simulation. When the main server receives a request for an upload service from the client, the main server requests an upload service to the others. If the upload service is completed, the client requests a compile service to be performed in the main server. The main server does the same procedure as the upload service. After finishing the compile request, the client sends a simulation request to the main server. These procedures are displayed by solid-line arrows among the components. This is the top layer of the DEVS simulation on SOA.

The main server generates and stores proxies of simulation services to which DEVS models are assigned as soon as the simulation request is received. Each simulation service holds an atomic model or atomic models on the storage. In the case of a coupled model, there is a mechanism of coupled model abstraction [36] to an atomic model with a DEVS state machine because there is no support of the

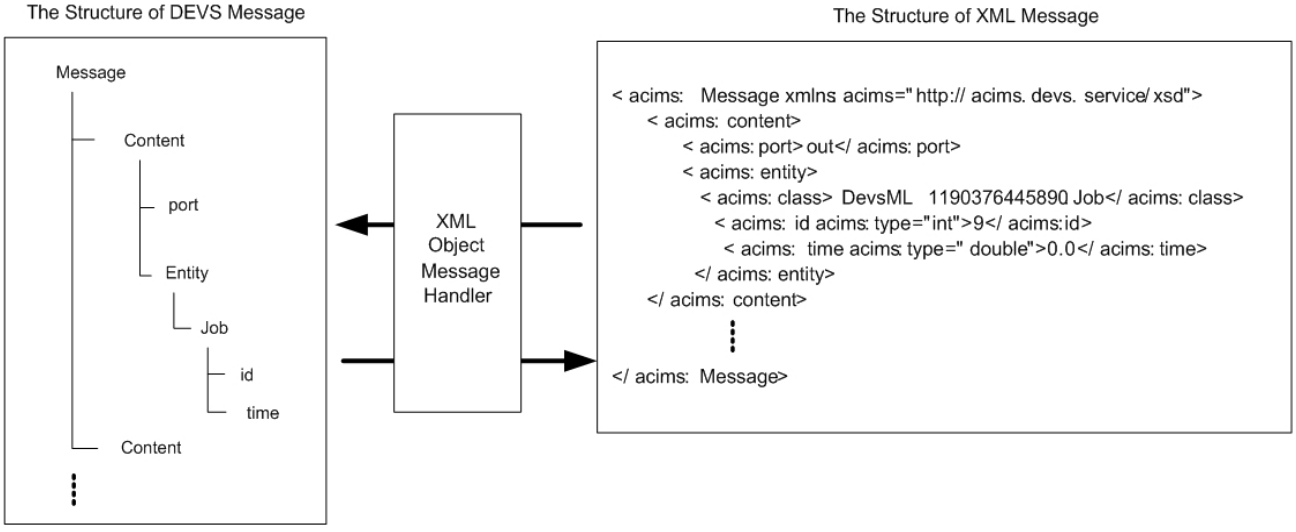

Figure 13. Example of XML object message handler

coupled simulation on the simulation service. Each simulation service sends messages to the main server encapsulating a coordinator according to the DEVS simulation protocols. This is a middle layer of the DEVS simulation on SOA, which is displayed by dotted-line arrows among the servers.

After the completion of the simulation, the client sends a request for the simulation results to the main server. In the DEVS simulation in this paper, a collector DEVS atomic model collects the simulation results sent from each DEVS model on each server. The main server sends the request for simulation results to the server possessing the collector DEVS model, receives the results, and sends the results to the client. This is a third layer of the DEVS simulation on SOA, which is displayed by dashed-line arrows between the client and the main server.

In this version of DEVS simulation on SOA, the client has equipment for displaying the simulation results in graphic charts. The results are stored in a file named *result.txt*, processed to data format, which the charts use as an input.

The upload of models is done through serialization and SOA technologies, and message passing is done through XML style message and SOA technologies. Figure 13 is an example of a DEVS message to an XML-style message conversion. A DEVS message is a language-specific object class, and the web service does not have an apparatus to send an arbitrary object message to another service because the web service supports only fixed structured messages defined in the WSDL. A DEVS message is too dynamic to be defined as one type of class in the WSDL. So, an XML object message handler is employed to transform an object DEVS message to an XML-style message. As seen in Figure 13, the structure of the DEVS message consists of at least more than one contents containing a port and Entity object. Entity objects can be any type of object inherited by the Entity. This DEVS message is converted to an XML-style message by the XML object message handler.

The DEVS simulation on SOA is a centralized simulation done through a central coordinator, which is located at the main server. Simulation begins with the coordinator requesting *nextTN* to all simulation services. After receiving all responses from all simulation services, the coordinator sends *minTN* to all simulation services. If any simulation service matches with *minTN*, the simulation service produces an output message propagated to the coordinator and sent to a simulation service or simulation services according to the coupling information. The output message is an XML-style message produced by an XML object message handler. After the message sending is finished, the simulation time is updated, and the coordinator requests a delta function to all simulation services. If there are some simulation services receiving a message from the external models, they execute the external transition function. After that, the coordinator repeats the above procedures until the simulation termination condition is met.

Figure 14 illustrates the DEVS simulation on SOA that is applied to a network behavior analysis example, which is the case when a client wants to analyze protocol uses and evaluate network throughput. There is a data extraction web service server inside a subnet. The server for a data extraction web service captures network behaviors and stores the network activities in a database. There are three servers: server 1 acts as a coordinator, server 2 analyzes protocol uses, and server 3 measures the network throughput out of the subnet. The four servers (one in

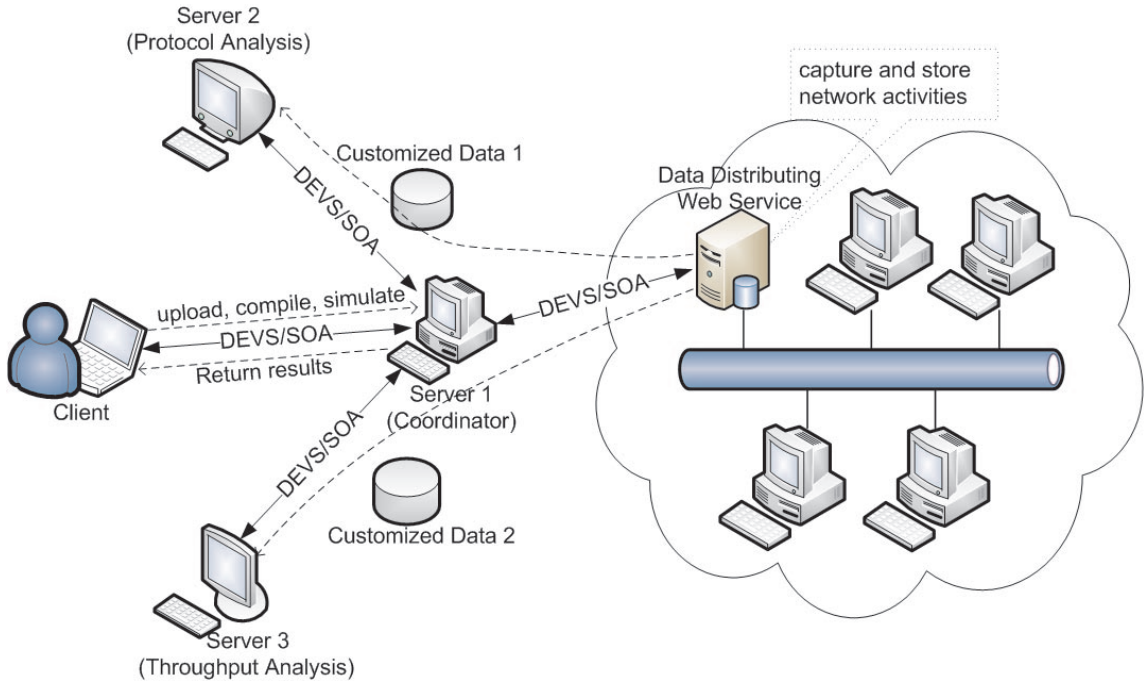

Figure 14. A network behavior analysis using DEVS/SOA

the subnet and three out of the subnet) are linked under the DEVS/SOA environment. The two servers (the protocol analysis server and the throughput analysis server) receive customized data for specific analysis from the data extraction server. The customized data are relatively size compared to the original data, which is stored in the data extraction server. Deploying workloads into multiple machines (assigning protocol analysis to server 2 and throughput analysis to server 3) reduces the computational burden of servers. Small-size customized data decreases communication overheads among servers and a small amount of data is effective in analyzing the data. These two factors, distributed workloads and small-size customized data, enable clients to obtain simulation results quickly and efficiently.

5.2 Real Time DEVS Simulation on SOA

The other approach of DEVS simulation on SOA is real-time simulation in which the next time for occurring internal transition passes by real time. Real-time simulation requires timely completion in physical time for the execution of a simulated model and it has been researched in various domain areas. Real-time DEVS (RT-DEVS) is employed to verify that the interactions among model components are correct in their relation to real time. Unlike virtual time simulation, time synchronizes simulation protocol to simulate DEVS models on SOA, and RT-DEVS simulation has the minimum network activity among simulators, because the simulators only invoke web services at the time of the propagation of out messages. In addition, it is a decentralized simulation because there is no coordinator to supervise all *RTSimulators*. Each *RTSimulator* follows a procedure to simulate their DEVS model without intervention for synchronization.

Figure 15 represents the overall structure of the RT-DEVS simulation system on SOA. As seen in the figure, each server participating in the simulation has two web services similar to centralized simulation. However, some functions in the simulation service, and classes such as the *RTCoordinator* and the *RTSimulator*, are added to support real-time simulation. The *RTCoordinator* used in the *MainService* and the *RTSimulator* used in the *Simulation* are made of multi-threads. The *RTCoordinator* generates proxies for *Simulation* services with DEVS models and coupling information that contains port names and addresses in which DEVS models are placed, and runs the *RTSimulators* in the *Simulation* services. Real-time simulation begins with a client program, such as centralized simulation on SOA. The solid lines in Figure 15 represent uploading files, compiling the files on each server, and executing *RTSimulators* on *Simulation* services. The dashed lines show out-message passing routes.

Figure 16 depicts RT-DEVS simulation protocol. The protocol starts with the initialization of the DEVS models in the *RTSimulators*. Each *RTSimulator* waits for pass-

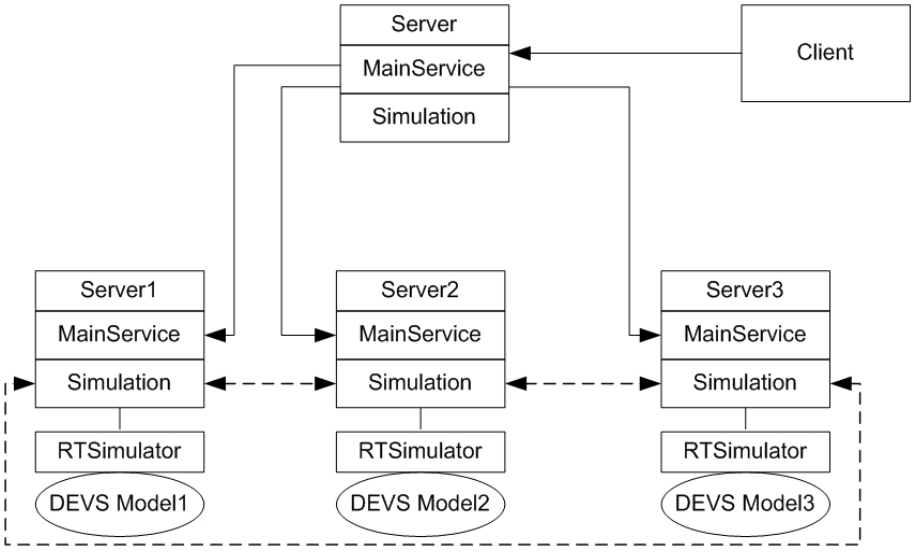

Figure 15. Overall architecture of RT-DEVS simulation system on SOA

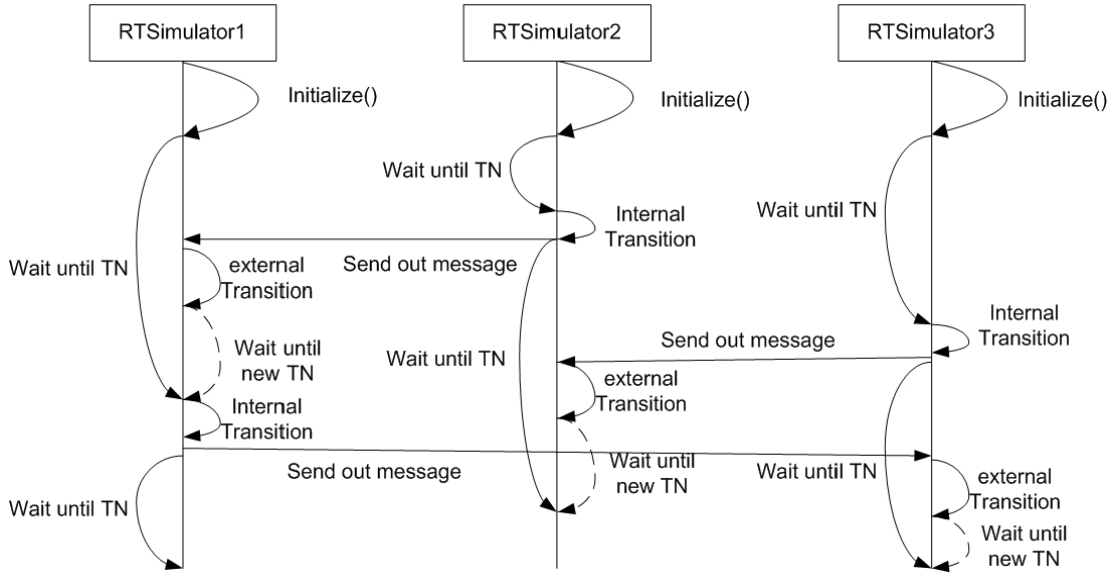

Figure 16. Real-time simulation protocol

ing event time (*tN*), after which internal transition occurs. If one of the *RTSimulators* has wall-clock time equal to *tN*, the *RTSimulator* executes an internal transition function consisting of the *lambda* function, which produces an out message, the propagation function, which sends the out message to other *RTSimulators* according to coupling information, and the delta function, which handles internal and external events. *RTSimulator2* in Figure 16 shows 'send out message' after internal transition and wait again with *tN* regenerated by the *delta* function. Meanwhile, *RTSimulator1* receives a message from the *RTSimulator2*, executes the external transition function having the delta function, and recalculates *tN* to wait. The interaction between *RTSimulator2* and *RTSimulator1* does not affect *RTSimulator3*. The way to influence other simulators is by sending messages.

Although RT-DEVS simulation has minimum network traffic, in the case of network delay and a tiny value of *tN*, the simulation might fail to get the correct results because of the distorted protocol. To filter the problem, it is impor-

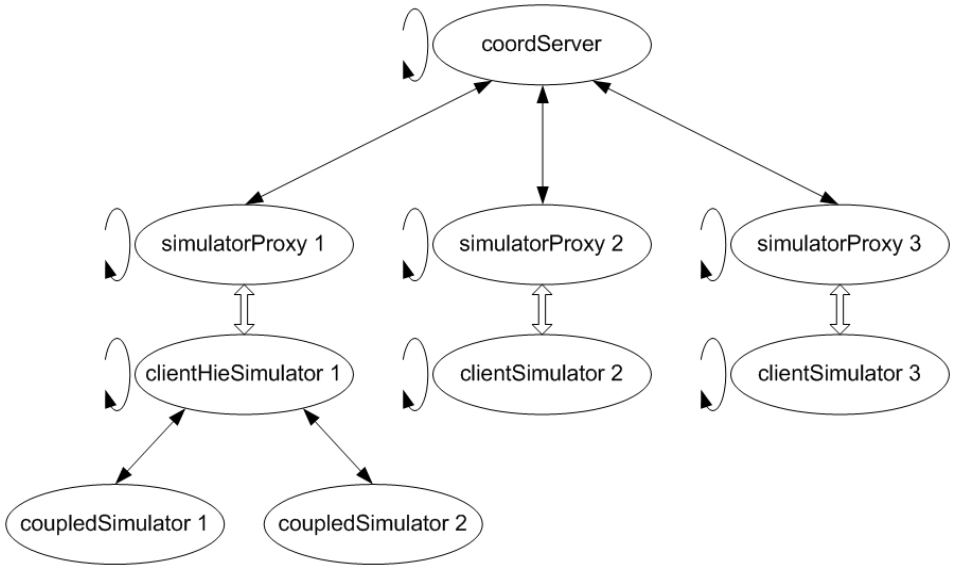

Figure 17. Distributed DEVS simulation

tant to know the threshold value of *tN* in order to for the RT-DEVS simulation to complete or speed up. However, it is difficult to select a safe threshold value of *tN*, since it is dependent on simulation environments, such as system performance, network throughput, etc.

6. Distributed Discrete Event System Specification Models and Simulators

A distributed SES/NZER is different to classic single-machine DEVS simulation. In this section, we illustrate how DEVS models, which are deployed in multiple machines in networks, can be simulated. Distributed DEVS models have components (DEVS atomic models and DEVS coupled models) of a DEVS coupled model that are distributed on several host computers. Figure 17 shows the distributed DEVS simulation that applies to both virtual-time and real-time simulations.

For distributed DEVS simulation, there must be a controller, a *coordServer*, which manages a whole simulation cycle and synchronizes all of the distributed simulators. The *coordServer* is responsible for passing messages among distributed simulators, as well as for advancing the DEVS models that are dispersed in the networks. The *coordServer* could be in a host that also holds a distributed simulator, or the *coordServer* could stay on an independent machine. Distributed machines, which include DEVS atomic models or DEVS coupled models, need simulators, *clientSimulators* for atomic models or *clientHieSimulator* for coupled models, on the machines. The *clientSimulator* is responsible for simulating a local DEVS atomic model. The *clientHieSimulator* is responsible for simulating a local DEVS coupled model, and there is a *coupled-Simulator* to take care of a local DEVS atomic model. The *coordServer* creates *simulatorProxys* that facilitate the *coordServer* communicating with the corresponding *clientSimulators* or *clientHieSimulators*. In addition, all of the distributed components, the *coordServer*, the *simulatorProxys*, the *clientSimulators*, and the *clientHieSimulators*, have their own thread. Figure 18 shows an example of DEVS modeling for a *SimForNTA*.

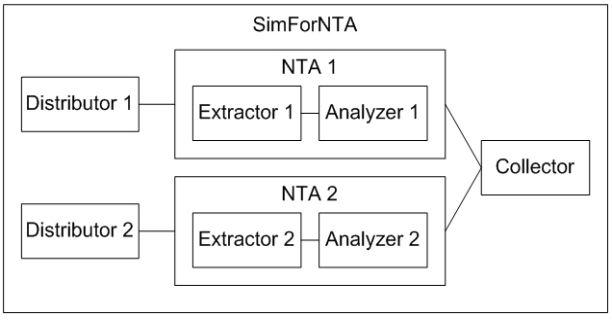

Figure 18. DEVS modeling: *SimForNTA*

The top level of a coupled model is a *SimForNTA*. The *SimForNTA* is composed of two coupled models, *NTA 1* and *NTA 2*, and three atomic models, *Distribute 1*, *Distributor 2*, and *Collector*. Two sub-coupled models (*NTA 1* and *NTA 2*) include their own components (an *Extractor* and an *Analyzer*). To achieve a fast analysis time, we apply the D&C approach. The whole job is divided by two, and each divided work is assigned to different processors. The *Distributor 1* and the *NTA 1* evaluate one half of the

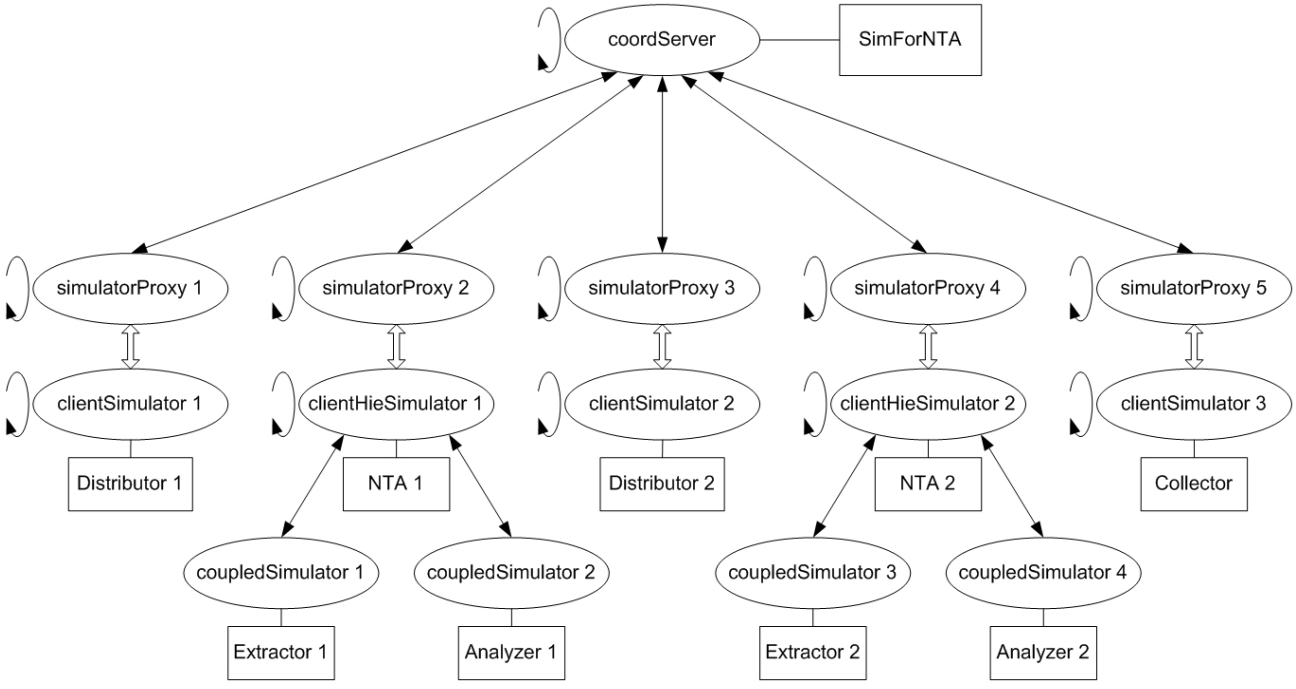

Figure 19. Distributed DEVS simulators and models for *SimForNTA*

whole work, and, at the same time, the *Distributor 2* and the *NTA 2* examine the other half of the whole job. Subsequently, the *Collector* model gathers the analyzed results from the two processes. We assign all of the models to different computers that are connected in the networks. Figure 19 illustrates a hierarchically structured distributed DEVS simulator and the corresponding DEVS models.

In this example, the top level coupled model, the *SimForNTA*, two sub-coupled models, *NTA 1* and *NTA 2*, and three atomic models, *Distributor 1*, *Distributor 2*, and *Collector*, are distributed into six computers. The *coordServer* for *SimForNTA* creates five *simulatorProxys*. Each *simulatorProxy* helps the *coordServer* to communicate with its corresponding *clientSimulator* or *clientHisSimulator*. In distributed DEVS simulation, the top-level coupling information is kept by the *coordServer*. The coupling information is downloaded to each *simulatorProxy*, and each *clientSimulator* or *clientHieSimulator* does not know the coupling information. The coordinator controls a whole simulation cycle and helps to pass messages among *clientSimulators* or *clientHieSimulators*. If the *Distributor 1* wants to send a message to the *NTA 1*, the *clientSimulator 1* sends the message to *simulatorProxy 1* over networks. Consequently, the *coordServer* decides the target host according to the top-level coupling information and puts the message to the *simulatorProxy 2*. Finally, the message is delivered to the *NTA 1* in the *clientHisSimulator 1*. Sending messages among DEVS models in a distributed computer requires network communication overheads. However, each *clientHieSimulator* keeps its local coupling information. As a result, messages are transmitted directly among *coupledSimulators*, not through *simulatorProxys*. For example, if the *Extractor 1* needs to send a message to the *Analyzer 1*, the *coupledSimulator 1* puts the message directly to the *coupledSimulator 2*. Therefore, there are no network communication overheads in this case.

Although *coordServer*, *simulatorProxys*, *clientSimulators*, and *clientHieSimulator* have their own thread, the slowest thread determines the overall simulation speed in the D&C mechanism, because the D&C is a pipeline with a divider, processors (in parallel), and a compiler, so the slowest one of these stages determines the overall speed. Therefore, speeding up all of the threads is important and reducing the communication overhead over networks is also a critical issue in distributed simulation environments.

Even though a distributed SES/NZER follows a decentralized distributed DEVS simulation scheme, couplings among components (DEVS atomic models and DEVS coupled models) keep the function as a single-machine DEVS. For example, a coupled model, *coupledModel*, is composed of three atomic models: *atomicModel 1*, *atomicModel 2*, and *atomicModel3*, so we could assign *atomicModel 1* to host 1, *atomicModel 2* to host 2, *atomicModel 3* to host 3, and the *coupledModel* to host 4 or one of the hosts that hold the atomic models. Therefore, the *coupledModel* controls synchronization among the atomic

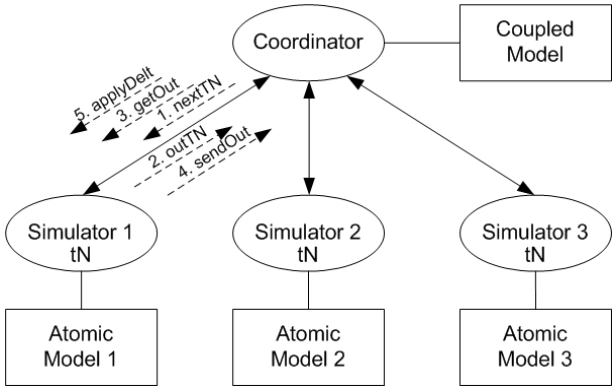

Figure 20. Basic DEVS simulation protocol

models. There must be message transmissions to control a whole DEVS simulation cycle.

The most considerable factor in distributed simulation over the web is how to reduce communication overheads. A distributed SES/NZER is performed under loosely coupled environments over the web and the DEVS is used for the simulation engine. To advance the simulation cycle, the basic DEVS simulation protocol requires five message transmissions, *nextTN*, *outTN*, *getOut*, *sendOut*, and *applyDelt*, among a coordinator and simulators. The DEVS protocol is described below and in Figure 20:

The coordinator sends a *nextTN* message to request the next event time (*tN*) from each of the simulators.
All of the simulators reply with their *tNs* in an *outTN* message back to the coordinator.
The coordinator sends to each simulator a *getOut* message containing the global *tN* (the minimum of the *tNs*).
Each simulator checks if it is imminent, which means its *tN* is equals to the global *tN*, and if so, returns an output of its model in a message to the coordinator in a *sendOut* message.
The coordinator uses the coupling specification to distribute the outputs as accumulated messages back to the simulators in an *applyDelt* message to the simulators. For those simulators not receiving any input, the messages sent are empty.

The basic DEVS simulation protocol is illustrated in Figure 20. If a coupled model and all of the atomic models are assigned in different machines that are connected in networks, the DEVS protocol overheads may exceed the advantage of the distributed simulation deploying workloads. Diminishing the number of DEVS protocol messages among computers results in decreasing communication overheads. Therefore, we may expect an overall speed up. In an effort to reduce DEVS protocol overheads, we apply two approaches: closure under coupling and minimizing the number of states. The closure under coupling allows us to use the networks of systems as components in

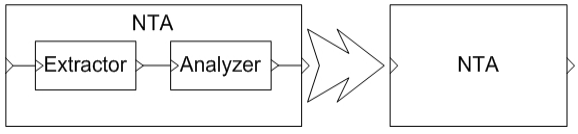

Figure 21. Closure under coupling

a larger coupled system, leading to hierarchical, modular construction [1]. This means that every coupled model is behaviorally equivalent to a basic atomic model.

Figure 21 presents the closure under coupling. The coupled model NTA is composed of two atomic models: the *Extractor* and the *Analyzer*. The closure under coupling makes these three DEVS components become one component, the NTA atomic model. We translate the coupling information of the coupled model, NTA, into a flat-structured atomic model, NTA. By this translation, the hierarchical structure of the DEVS model can be flattened. Message exchanges consume a large amount of time if the model structure is complex in distributed environments. On the other hand, if the model hierarchy is flattened, communication overheads among models can be minimized. Therefore, a flat-structured modeling approach facilitates reducing the number of messages, and we can achieve better performance results [38, 39]. In DEVS/SOA environments, a *coorServer* creates *simulatorProxys* – as many as the number of total models. Even though, the coupled model, NTA, and two atomic models, the *Extractor* and the *Analyzer*, are assigned into one computer with single IP address, a *coordServer* creates three *simulatorProxys*. Therefore, the *coordServer* needs more processing time to decide a destined *simulatorProxy* among three *simulatorProxys* for a message. If the atomic model NTA replaces the three component DEVS models, only one *simulatorProxy* is created by the *coorServer*. As a result, we could obtain a speed up. Figure 22 shows that the closure under coupling decreases the number of *simulatorProxys* and simplifies the DEVS simulation architecture. Figure 22(a) illustrates a simulation environment before DEVS models are refined, and the right figure presents the refined DEVS model.

In addition to the effort of reducing the number of DEVS models (atomic models and coupled models), we decrease the number of state transitions in atomic models. For each simulation cycle, there are five message transmissions between a *coordServer* and *clientSimulators* or *clientHieSimulators*. The processing time for these DEVS protocol message transmissions should not overwhelm the processing time of the processor. An atomic model of the SES/NZER loads PES XML documents and analyzes one tuple of information at one state transition. This approach needs many state transitions according to the number of tuples in the PES XML files. For example, there are ten PES XML files, and each PES XML file includes 1,300

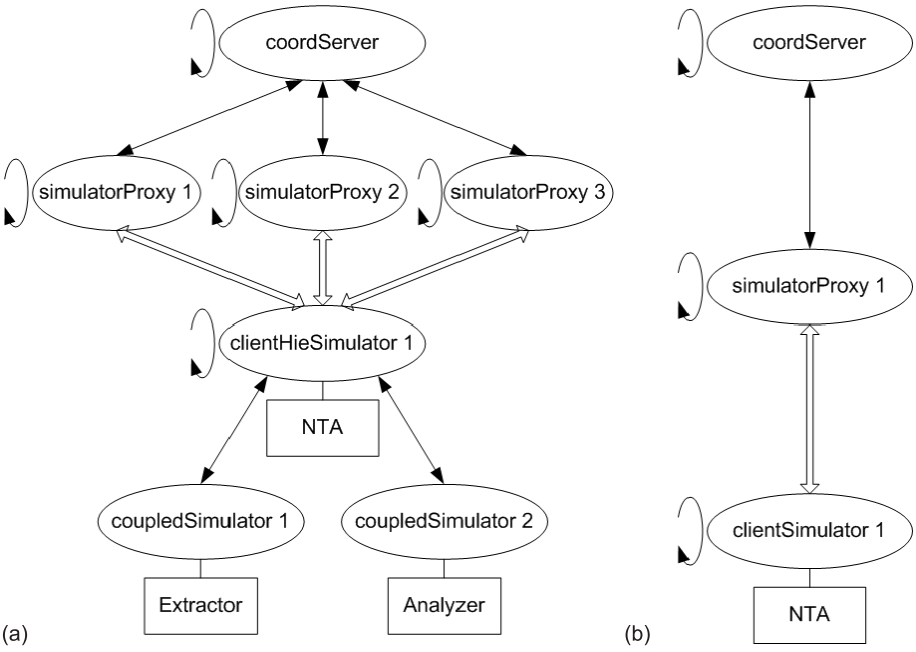

Figure 22. DEVS model comparison under the DEVS/SOA environment

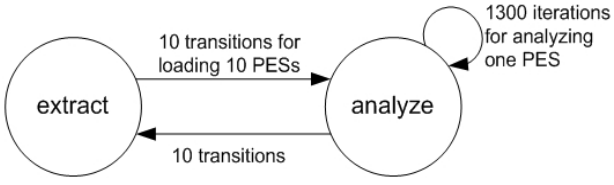

Figure 23. State transition diagram in the SES/NZER

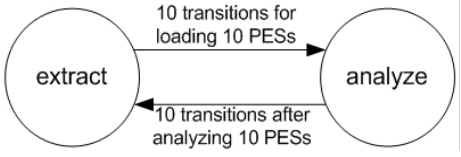

Figure 24. State transition diagram in the distributed SES/NZER

tuples of information. Then, there must be 13,020 state transitions. The 13,020 transitions include 10 state transitions (the *extract* state to the *analyze* state) after loading the PES XML documents, 1,300 × 10 iterative transitions (the *analyze* state to the *analyze* state) for evaluating all of the tuples in the 10 PESs, and 10 transitions (the *analyze* state to the *extract* state) to load the PES files. Figure 23 shows these state transitions.

A coordinator sends and receives a total of 65,100 (5 × 13,020) message transmissions only for DEVS protocol processing. Although the size of a DEVS protocol message is trivial, 65,050 message transmissions is a considerable number. For distributed simulation, if workloads are distributed to five computers, the total number of DEVS protocol messages is 325,500 (5 × 65,100). In this case, the communication overhead is too great for only advancing simulation cycle. So, we fit the SES/NZER's atomic models to a distributed simulation. A NTA atomic model of the distributed SES/NZER loads PES XML files and evaluates a complete PES document at one state transition. Therefore, the total number of state transitions in this example is 20. The 20 transitions include 10 state transitions (the *extract* state to *analyze* state) for loading 10 PES files and 10 state transitions (the *analyze* state to the *extract* state) after examining all 10 datasets. Figure 24 illustrates an updated state transition diagram for the distributed SES/NZER.

Reducing the number of state transitions results in decreasing communication overheads, which are caused by passing DEVS protocol messages. Respectively, we could speed up the overall simulation time over network environments. Lee's PhD dissertation [40] discusses the effect of quantization in distributed DEVS/High-level Assembly (HLA) environments. In addition, communication latency and an overhead reduction technique in distributed interactive simulation are introduced through an approach of bundling the Protocol Data Unit (PDU) [41].

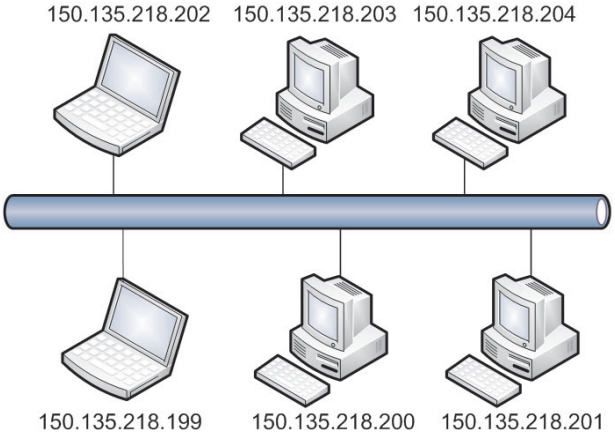

Figure 25. Testbed for distributed simulation using DEVS/SOA

7. Experimental Results

We set up a testbed for a distributed simulation environment in the ACIMS laboratory as shown in Figure 25. We installed Apache Tomcat 6.0 on six computers (four desktop computers and two laptop computers) with the Windows XP operating system. Apache Tomcat is a servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies [42]. We installed an Apache Axis2/Java web service engine [43]. Apache Axis2 is the core engine for web services, and it is an implementation of the World Wide Web Consortium (W3C) SOAP.

We monitored and captured network activities inside the ACIMS laboratory subnet and we used the captured data for generic network behavior analyses, such as protocol evaluation and throughput measurement. For the intrusion detection analyses, the KDD'99 dataset was used as source data.

7.1 Network Traffic Analysis

This section presents the experimental results for a generic network behavior analysis. We preset two analyses, protocol and throughput analyses, in a user's request input system (shown in Figure 26). According to target analyses, the corresponding required attributes are selected automatically. Alternatively, users could choose attributes if they wanted to evaluate their specialized target analyses. Target analyses selections generate new SESs. The newly generated SESs act like agents, so overall simulations are controlled by these new SESs. The deciding time frames are next and, finally, the customers select the degree of parallelism, which is the number of computers for distributed simulations. Requests, which are combinations of target time frames and the number of simulation machines, create new DEVS coupled models. Data is partitioned by the number of hosts, and each portion of the data is assigned to corresponding computers. A simulation model partitioning approach in distributed simulation is proposed and implemented in Zhang's PhD dissertation [44]. The next step is assigning DEVS models into distributed computers. Once a top-level coupled model is selected, this selection holds the top-level coupled model's following child components. After allocating models into dispersed machines, a simulation starts to examine users' requests. Figure 26(b) shows the processes of choosing a top-level coupled model and assigning models into distributed servers.

Table 4. Data size comparisons for NTAs

Data	Original	PES for protocol	PES for throughput
Half day	2.83 GB	168 MB	200 MB
One day	5.44 GB	326 MB	387 MB
Two day	10.8 GB	646 MB	770 MB

The original data sizes for half a day, one day, and two days are 2.83, 5.44, and 10.8 GB. Instead of keeping all of the attributes, the PES XML documents for protocol analysis hold two attributes: a packet ID and a protocol type. So, the PES file sizes are 168, 326, and 646 MB. Their sizes are about 6% of the original data size. The PES files for throughput evaluation include two attributes, an event time and a packet size, and their sizes are 200, 387, and 770 MB. The ratio is about 7%. Table 4 presents the data size comparisons between original data and the PES data for the NTAs.

In addition to measuring the data size, we examined the execution times of half a day, one day, and two days of data of both the protocol analysis and the throughput measurement by varying the degree of parallelism (number of computers for analysis). We experimented with four sorts of server sets: a local machine, two machines (one distributing server and one analyzing server), four machines (two distributing servers and two analyzing server), and six machines (three distributing servers and three analyzing servers). The execution time was composed of three sub-times: the time for distributing the data to the servers, the time for evaluating the received data at the analyzing servers, and the time for collecting and displaying the evaluated results at a client computer.

For three different datasets, we measured three kinds of times: the distributing data time, the analyzing data time, and the collecting resulting data time. We measured the execution times at four different sets of computers: a local computer, one distributing data computer and one analyzing data computer, two distributing data computers and two analyzing data computers, and three distributing data computers and three analyzing data computers. We notice that the distributing times increase gradually as the number of distributed computers increases. Ideally, distributing times must decrease in the counter ra-

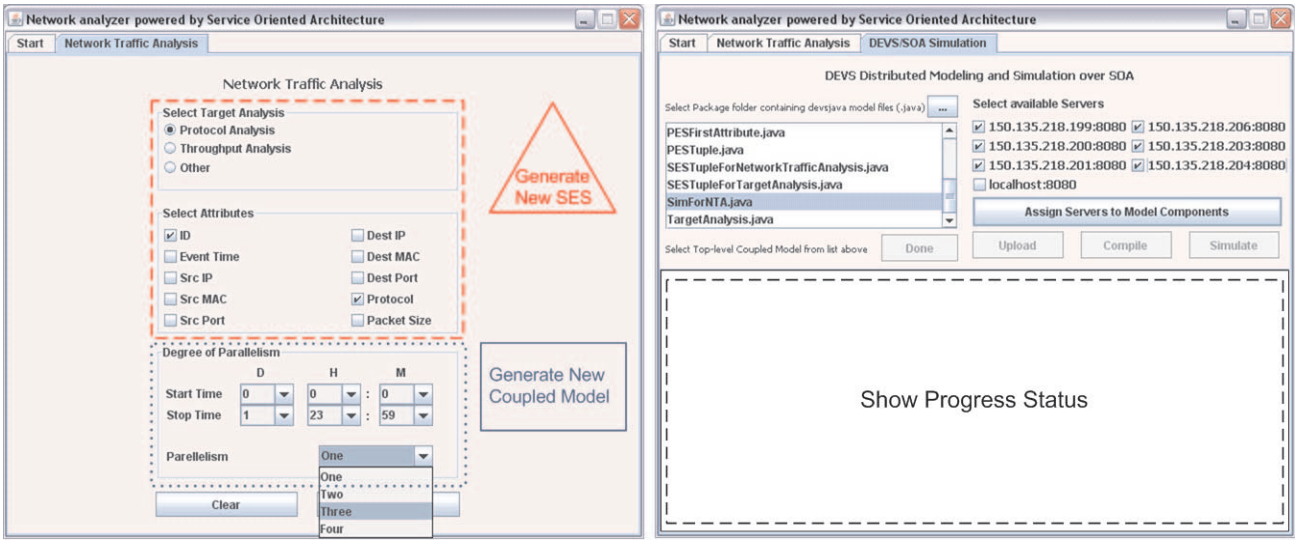

Figure 26. Snapshot of the distributed SES/NZER

Table 5. Execution time comparisons between virtual-time and real-time simulations

Analysis	Protocol analysis		Throughput analysis	
Simulation time	Virtual time	Real time	Virtual time	Real time
Distributing time	33 min 21 s	7 min 37 s	42 min 14 s	10 min 4 s
Analyzing time	3 min 56 s	36 s	3 min 58 s	37 s
Collecting time	2 s	1 s	1 s	1 s
Total	37 min 19 s	8 min 14 s	46 min 13 s	10 min 42 s

tio of the number of hosts. However, communication overheads (data messages and DEVS protocol messages) prevent us from achieving optimal results. We see that analyzing data times are reducing as the number of computers is increasing. Unlike distributing times, analyzing times are not affected by network communication overheads. Because collecting resulting data times are one or two seconds in most cases, we could forgo the collecting times for comparing execution times. We also experimented with real-time simulation. Because each simulator in each different machine has its own simulation time, and the overall execution time is not affected by the communication overheads that are caused by the DEVS protocol messages and data messages between a centralized coordinator and distributed simulators, we achieve speed up when comparing to virtual-time simulation.

Table 5 shows execution time comparisons between virtual-time simulations and real-time simulations. In the virtual-time DEVS/SOA simulation, all of the simulation servers are controlled by a top-level coordination server for advancing discrete events and passing messages among simulation servers, even though each simulation server runs by itself and does not affect the other simulation servers. This is a centralized approach, and this simulation causes time delay. The overall simulation speed fits to the slowest server's evaluating time. In addition, there must be many sets of message transmissions, *nextTN*, *outTN*, *getOut*, *sendOut*, and *applyDelt*, between a top-level coordinating server and the model simulating servers for the DEVS protocol. These DEVS protocol messages are another cause of degrading simulation speed. To overcome these limitations of virtual-time simulation, RT-DEVS/SOA simulation is applied, and, finally, we accomplish the goal of distributed simulation, and speed up execution times, through real-time simulation. Figure 27 illustrates the real-time simulation results for both protocol and throughput analyses.

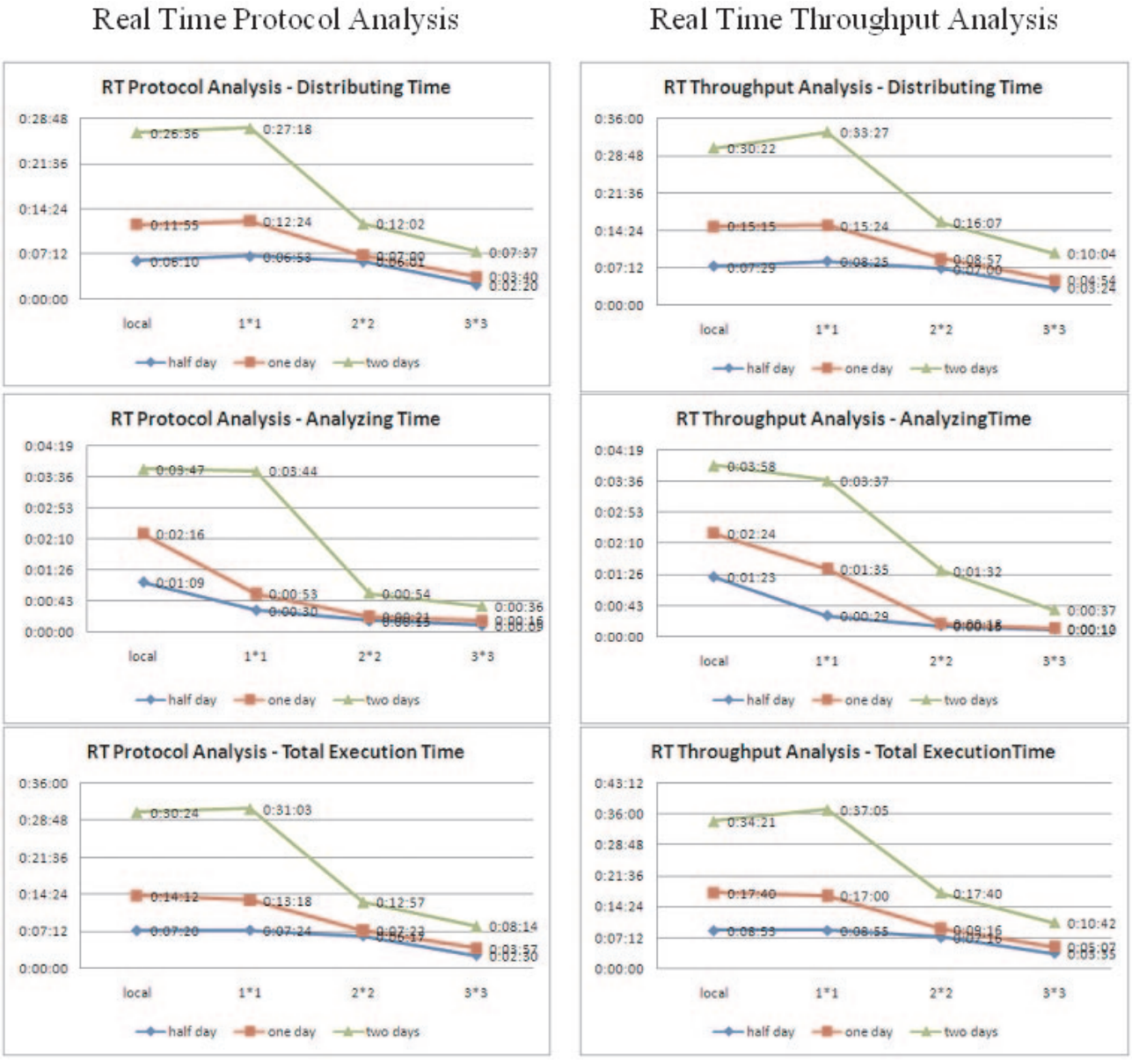

Figure 27. Real-time simulation results for network behavior analyses

7.2 Evaluations of Intrusion Detection Systems

Recall that we built two IDS agent models: the LAND agent and the POD agent. As illustrated in Section 7.1., after customers' requests, which select a target IDS, the time frames (start time and end time) and a degree of parallelism (the number of distributed computers for analysis) are applied through an input system. Users could assign simulation models into multiple servers according to the selected degree of parallelism. Firstly, we measured the data sizes. The original source data size for two weeks (five days a week) is 4.12 GB. The pruned data size for the LAND IDS, which includes even times, source host IP addresses, and the destination host IP addresses size, is 368 MB. The data size for the POD IDS is 437 MB. These PES data sizes are about 9%t (LAND) and 10% (POD) of the original KDD'99 dataset. Table 6 presents the data size comparisons for IDS evaluations.

In addition, we observed the IDS evaluating times of both the LAND attack and the POD attack using the two weeks of the KDD'99 dataset. We differentiated the number of computers as for the experiment for generic NTA.

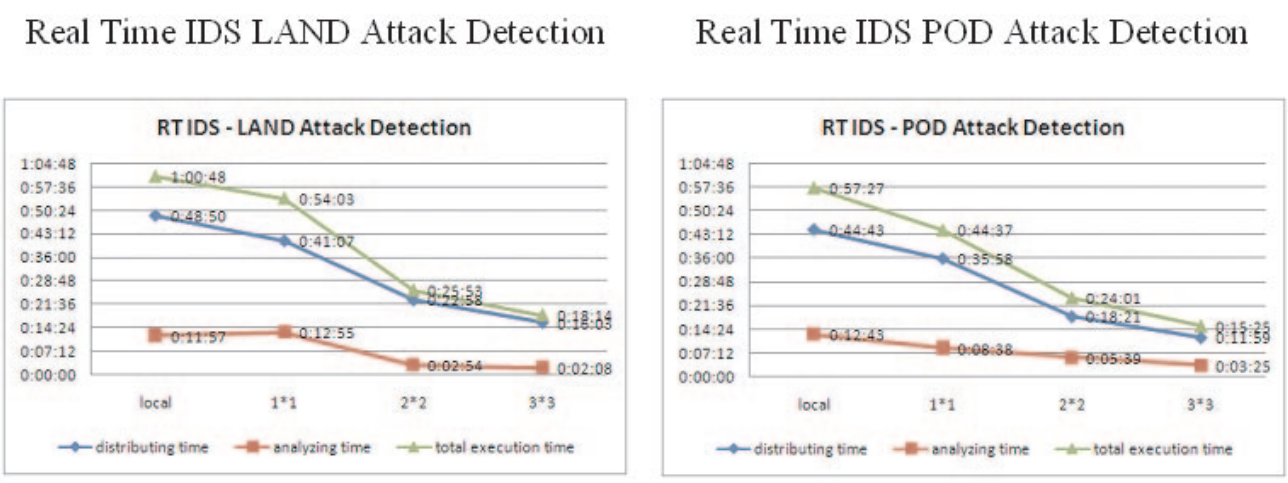

Figure 28. Real-time experimental results of IDS analyses

Table 6. Data size comparisons for IDS evaluations

	Original	PES for LAND	PES for POD
Source data (2weeks)	4.12 GB	368 MB	437 MB

We achieved similar execution times to those in Section 7.1. In the virtual-time simulation, we noticed that distributing data times are increased as the number of evaluating machines increased, which is due to overheads (network packet transmission delays and DEVS protocol message overheads). Next, we noticed that the analyzing times are increased. In the real-time simulation, both the distributing times and the analyzing times decreased as the number of computers increased. Therefore, we achieved fast total execution times in the real-time simulation. The experimental results of the real-time simulation of IDS analyses are presented in Figure 28.

Figure 28 illustrates that we achieved a speed up of the total execution times in the real-time simulation. The experimental results in this section show that a distributed SES/NZER reduces data sizes in terms of different customers' requests in both virtual-time and real-time simulations. In addition, a distributed SES/ZER speeds up analyzing times by dividing a whole workload into several small jobs and deploying the small jobs into multiple machines. In addition, we achieved fast execution times in real-time simulations, since real-time simulations reduce the message transmission delay overheads that occur in virtual-time simulations.

8. Discussion

This study proposes a web-based distributed simulation for NTAs over a SOA. The main objective of this study was to develop an approach for quick and efficient network behavior analysis. To deal with large numbers of network behaviors being quickly and efficiently analyzed, the SES theory was applied. The SES facilitates implementing a system to achieve our main goal. The SES is a theory for designing structured information hierarchically and efficiently. Specifically, the SES is very useful for data engineering. We designed a generic network behavior in SES format. We must notice that every customer has different requests (different applications). For example, some customers want to evaluate network protocol uses. On the other hand, some users want to measure network throughput. Depending on various requirements (pragmatic frames), systems need to be optimized for the pragmatic frames to speed up analysis time effectively. Two processes that create a new SES to correspond to users' requests and enable systems to be adaptively optimized are pruning operations and mapping the newly generated SES with the pre-defined SES, which represents a generic network packet behavior. Reactions to pragmatic frames facilitate systems keeping accurate data only, so we are able to reduce overall data size. Therefore, we could analyze extensive long-term network activities, which Ethereal cannot do. Although we enabled large amounts of data to be examined, we still needed a long evaluation time. To speed up the evaluation time, we applied a web-based distributed simulation approach over a SOA. Deploying workloads into multiple machines decreases the burdens of individual computers, and results in the hosts, which have low computational powers (CPU and memory), participating in large-scale simulations. As a result, there is no longer any need for super computers. The DEVS/SOA facilitates deploying workloads into multi-servers, and, consequently, increasing overall system performance.

In this study, we built two IDS agent models, the LAND attack agent and the POD attack agent, and evalu-

ated the two models. One advantage of the SES/NZER is that it provides a simulation framework for testing IDSs. The SES/NZER is available for IDS researchers to test their algorithms. IDS researchers build their models corresponding to their IDS algorithms, and they request the necessary attributes to evaluate their models. Other required models for simulations are provided. In addition to this scalability, the SES/NZER should include more pre-defined models, which are agents to detect various intrusions. Intrusions are classified into five kinds: DoS attacks, User to Root attacks, Remote to Local attacks, Probe attacks, and Data attacks. If the SES/NZER were capable of more functions, it could give more convenience to users as a concrete tool. Intrusion detection algorithms should reserve specific policies. Each attack signature (attack detection policy) needs a different set of information to detect a corresponding attack. If IDS developers want to examine whether their IDS algorithms work well, the necessary attribute values in the network packet headers must be provided. According to researchers' target IDS algorithms, new SESs, which represent the required attributes, have to be generated, and, subsequently, the new SESs are used for pruning entities and mapping to the generic network behavior SES, which is described in Figure 7. For example, in detecting an Apache2 attack it is necessary to scrutinize whether the packet headers with HTTP *GET* requests with the header 'User-Agent: sioux\r\n' are over a certain number [35]. A typical HTTP request contains 20 or fewer headers in most systems. Therefore, a corresponding SES must hold three entities: the protocol type, the source IP address, and the packet header information. Similar to this Apache2 attack example, new SESs are generated when researchers ask to analyze the other intrusions. In addition to these specific IDS cases, general cases must be covered too, because new intrusions are being created constantly. To achieve accurate results for both non-specified general analyses and totally new attacks, we need to extend the generic network behavior SES shown in Figure 7 by including more entities, such as the Internet Header Length (IHL), the Type of Service (TOS), the Time to Live (TTL), the header checksum, and other obtainable attributes from the packet headers, into the SES. As a result, IDS developers may have better opportunities to evaluate precisely their algorithms.

9. Conclusion and Future Works

Recently, network uses have been increasing rapidly. Therefore, the size of data, which is caused by network activities, is getting larger. Network administrators or managers need NTA tools that can produce results quickly and accurately. There are several NTA tools, such as tcpdump, Ethereal, and other applications. However, these tools have drawbacks, namely limited data size and complications (large system memory and huge computational power requirements). In addition to these problems, the currently existing tools are limited to performing inside networks, due to security issues. The dump files that are monitored and captured by these tools include secure information, such as user IDs, passwords, and other information. These secure attributes must be protected against abnormal accesses, so observing network activities from outside the networks should be prohibited. However, network behaviors need to be analyzed outside the target networks in some cases.

This paper presents an approach to efficiently and quickly analyzing network behaviors by applying SES theory. We achieved both the evaluation of a large amount of network traffic activity data and the performance of a web-based distributed simulation over a SOA. In addition, we accomplished fast execution times through real-time decentralized distributed simulation. However, there are further research works: developing web services for NTAs and implementing additional attack-detecting functions for IDSs. The ultimate goal is to implement network behavior analyses web services. This study aimed for a decentralized distributed DEVS simulation to speed up evaluation times by deploying workloads into multicomputers. However, customers are still responsible for building models for simulating their systems. Web services, which are implementations of integrating an automated model constructing process with analyzing the corresponding system process, provide more accommodation to users. Another future work is implementing web service systems that will perform analyses of customers' data. Customers may provide data to multiple web services asynchronously. Subsequently, web services evaluate received data and give evaluated results back to customers.

10. References

[1] Zeigler, B.P., T.G. Kim and H. Praehofer, 2000. *Theory of Modeling and Simulation*, 2nd edn, Academic Press, New York.

[2] Cho, Y.K., B.P. Zeigler, H.J. Cho, H.S., Sarjoughian. H., and Sen, S., 2000. Design consideration for distributed real-time DEVS. In *Proceedings of the AI and Simulation Conference*, Tucson, AZ.

[3] Zeigler, B.P. 1984. *Multi-Facetted Modeling and Discrete Event Simulation*, Academic Press, New York.

[4] Zeigler, B.P. and G. Zhang, 1989. The system entity structure: knowledge representation for simulation modeling and design. In L.E. Widman, K.A. Loparo, and N.R. Nielsen (Eds.), *Artificial Intelligence, Simulation and Modeling*, pp. 47–73, Wiley, New York.

[5] World Wide Web Consortium (W3C), eXtensible Markup Language (XML), 2008, http://www.w3.org/XML/

[6] Zeigler, B.P. and P.E. Hammonds 2007. *Modeling & Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange*, Elsevier.

[7] Champion, M., C. Ferris, E., Newcomer, E., and Orchard, D., 2002. Web Services Architecture, W3C.

[8] World Wide Web Consortium (W3C), Simple Object Access Protocol (SOAP), 2008, http://www.w3.org/TR/soap/

[9] World Wide Web Consortium (W3C), Web Service Architecture, 2008, http://www.w3.org/TR/ws-arch/

[10] World Wide Web Consortium (W3C), Web Services Description Language (WSDL), 2008, http://www.w3.org/TR/wsdl20-primer/

[11] Universal Description, Discovery and Integration (UDDI), 2008, http://uddi.xml.org/
[12] Service-Oriented Architecture (SOA), 2008, http://www.sun.com/products/soa/index.jsp
[13] Representational State Transfer (REST), 2008, http://rest.blueoxen.net/cgi-bin/wiki.pl
[14] R.T. Feilding. 2000. *Architectural Styles and the Design of Network based Software Architectures*, PhD Dissertation, UNIVERSITY OF CALIFORNIA, IRVINE, CA.
[15] Mittal, S., J.L. Risco-Martín and B.P. Zeigler. Implementation of formal standard for interoperability in M&S/systems of systems integration with DEVS/SOA, C2 Journal, Vol 3. No. 1, 2009.
[16] Mittal, S., J.L.R. Martin and B.P. Zeigler. 2009. DEVS/SOA: A cross-platform framework for net-centric modeling and simulation in DEVS unified process. *SIMULATION: Transactions of SCS*, 85(July), 419–450.
[17] Seo, C. and B.P. Zeigler 2009. Interoperability between DEVS simulators using service oriented architecture and DEVS namespace. In *A Joint Symposium DEVS Integrative M&S (DEVS) and High Performance Computing (HPC) Proceedings of the Spring Simulation Conference*.
[18] DEVSJAVA, 2009, http://www.acims.arizona.edu
[19] ADEVS: an open source C++DEVS Simulation engine, 2009, http://www.ornl.gov/∼1qn/adevs/index.html
[20] Seo, C. 2009. *Interoperability between DEVS Simulators using Service Oriented Architecture and DEVS Namespace*, PhD Dissertation, University of Arizona, Tucson, AZ.
[21] Tolk, A., C.D. Turnitsa, S.Y. Diallo, and Leslie S. Winters, 2006. Composable M&S web services for net-centric applications. *The Journal of Defense Modeling & Simulation*, 3(1): 27–44.
[22] Wutzler, T. and H.S. Sarjoughian. 2007. Interoperability among parallel DEVS simulators and models implemented in multiple programming languages. *SIMULATION: Transactions of SCS*, 83(June): 473–490.
[23] Yoo, T., H. Cho and E. Yücesan. 2009. Web services-based parallel replicated discrete event simulation for large-scale simulation optimization. *SIMULATION: Transactions of SCS*, 85(July): 461–475.
[24] Wainer, G.A., R. Madhoun, and K. Al-Zoubi. 2008. Distributed simulation of DEVS and Cell-DEVS models in CD++ Ssing web-services. *Simulation Modelling Practice and Theory*, 16(9): 1266–1292.
[25] Wainer, G.A., Liu, Qi, Chazal, J., Quinet, L., and Taore, M.K., 2008. Performance analysis of web-based distributed simulation in DCD++: a case study across the Atlantic Ocean. In *Proceedings of the '08 Spring Simulation Conference*, pp. 413–420.
[26] Puketza, N., Zhang, K., Chung, M., Mukherjee, B., and Olsson, R.A., 1996. A methodology for testing intrusion detection system. *IEEE Transactions on Software Engineering*, 22(10): 719–729.
[27] Puketza, N., Zhang, K., Chung, M., Mukherjee, B., and Olsson, R.A., 1997. A software platform for testing intrusion detection systems. *IEEE Software*, 14(5): 43–51, September/October.
[28] Bishop, M., Cheung, S., Wee, C., 1997. The threat from the net. *IEEE Spectrum*, 38(8): 56–63.
[29] Karatsuba, A. and Y. Ofman. 1963. Multiplication of multidigit numbers on automata. *Soviet Physics doklady*, 7(7): 595–596.
[30] Arizona Center for Integrative Modeling and Simulation (ACIMS), 2008, http://www.acims.arizona.edu/
[31] Ethereal, Network Protocol Analyzer, 2008, http://www.ethereal.com/
[32] The UCI KDD Archive, 1999, KDD 1999 Cup dataset, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[33] DARPA Intrusion Detection Evaluation, Lincoln Laboratory, Massachusetts Institute of Technology, 2008, http://www.ll.mit.edu/IST/ideval/index.html/
[34] KDD Cup 1999, http://www.sigkdd.org/kddcup/index.php?section=1999&method=info
[35] Haines, J.W., R.P. Lippmann, D.J. Fried et al. 2001. MIT Lincoln Laboratory, 2001. *1999 DARPA Intrusion Detection Evaluation: Design and Procedure*, DARPA Technical report, February.
[36] Mittal, S. 2007. *DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures*, PhD Dissertation, University of Arizona, Tucson, AZ.
[37] Mittal, S., J.L. Risco-Martin, and B.P. Zeigler. 2007. DEVS-based simulation web services for net-centric T&E. In *Proceedings of the Summer Computer Simulation Conference SCSC'07*, July.
[38] Glinsky, E. and G. Wainer. 2002. Definition of real-time simulation in the CD++ toolkit. In *Proceedings of the SCS Summer Computer Simulation Conference*, San Diego, CA.
[39] Kim, K., Kang, W., Sagong, B., Seo, H., 2000. Efficient distributed simulation of hierarchical DEVS models: transforming model structure into a non-hierarchical one. In *Proceedings of the 33rd Annual Simulation Symposium*, Washington DC.
[40] Lee, J.S. 2001. *Space-Based Data Management for High Performance Distributed Simulation*, PhD Dissertation, University of Arizona, Tucson, AZ.
[41] Vargas, J., DeMara, R.F., Georgiopoulos, M., Gonzalez, A.J., Marshall, H. 2004. PDU bundling and replication for reduction of distributed simulation communication traffic. *The Journal of Defense Modeling and Simulation*, 1(3): 171–183.
[42] Apache Tomcat, 2008, http://tomcat.apache.org/
[43] Apache Axis2 Web service engine, 2008, http://ws.apache.org/axis2/
[44] Zhang, M. 2007. *Toward a Flexible and Reconfigurable Distributed Simulation: A New Approach to Distributed DEV*S, PhD Dissertation, University of Arizona, Tucson, AZ.

Taekyu Kim *is a senior research at the Center for Modeling and Simulation Studies, Security Management Institute, Seoul, Korea. He received PhD in Electrical & Computer Engineering (ECE) at the University of Arizona. He holds an MS (2006) in ECE from the University of Arizona and a BS (2000) in Computer Science and Engineering from the Chung-Ang University, Seoul, Korea. His research interests include DEVS-based hybrid system modeling, model based system design, ontology methodology, data engineering, and Semantic Web.*

Chungman Seo *is a research engineer at the RTSync company and a member of the ACIMS. He received his PhD in ECE from The University of Arizona in 2009. His research interests include DEVS-based web service integration; DEVS/SOA-based distribution DEVS simulation, and DEVS simulator interoperability.*

Bernard P. Zeigler *is a Professor of ECE at the University of Arizona, Tucson and Director of the ACIMS. He is internationally known for his 1976 foundational text Theory of Modeling and Simulation, revised for a second edition (Academic Press, 2000). He has published numerous books and research publications on the DEVS formalism. In 1995, he was named Fellow of the Institute of Electrical and Electronics Engineers (IEEE) in recognition of his contributions to the theory of discrete event simulation.*