ONTOLOGY/DATA ENGINEERING BASED DISTRIBUTED SIMULATION OVER SERVICE ORIENTED ARCHITECTURE FOR NETWORK BEHAVIOR ANALYSIS

By

Taekyu Kim

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2008

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Taekyu Kim
entitled Ontology/Data Engineering based Distributed Simulation Over Service Oriented Architecture for Network Behavior Analysis
and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy in Electrical and Computer Engineering.

____ Date: 04/16/08
Bernard P. Zeigler

____ Date: 04/16/08
Salim Hariri

____ Date: 04/16/08
Roman Lysecky

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.
I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

____ Date: 04/16/08
Dissertation Director: Bernard P. Zeigler

STATEMENT BY AUTHOR

SIGNED: Taekyu Kim

ACKNOWLEDGEMENTS

I would like to express my truthful gratitude to my advisor Professor Bernard P. Zeigler for his mentoring, guidance, and endless support during the course of this research period. He also inspires me to expand knowledge in discrete event simulation and ontology/data engineering fields.

I also express my appreciations to the committee members Professor Salim Hariri and Professor Roman Lysecky for providing suggestions enhancing the content of this dissertation.

I would like to express thank to my colleagues at ACIMS lab, Chungman Seo, Hojun Lee, Saehoon Cheon, Moath Jarrah, Saurabh Mittal, Rajanikanth Jammalamadaka, and Dr. DH Kim.

I would like to thank to my father and mother. They always support and encourage me. I appreciate that my wisdom and diligence inherit from my parents.

Special thank goes to my wife, Jiyoung Kim, for her constant support and appreciation. I am indebted to her for all the encouragement. Finally, my two daughters, Hannah Kim and Sheena Kim, make me never give up anything in my life.

TABLE OF CONTENTS

TABLE OF CONTENTS - *Continued*

TABLE OF CONTENTS - *Continued*

TABLE OF CONTENTS - *Continued*

LIST OF FIGURES

LIST OF FIGURES - *Continued*

LIST OF FIGURES - *Continued*

LIST OF FIGURES - *Continued*

LIST OF TABLES

ABSTRACT

As network uses increase rapidly and high quality-of-service (QoS) is required, efficient network managing methods become important. Many previous studies and commercial tools of network management systems already exist. But, these tools such as tcpdump, Ethereal, and other applications have weaknesses: limited size of files, command line execution, and large memory and huge computational power requirements. Researchers struggle to find fast and effective analyzing methods to save maintenance budgets and recover from systematic problems caused by the rapid increment of network traffic or intrusions. The main objective of this study is to propose an approach to deal with a large amount of network behaviors being quickly and efficiently analyzed. We study an ontology/data engineering methodology based network analysis system. We design a behavior, which represents network traffic activity and network packet information such as IP addresses, protocols, and packet length, based on the System Entity Structure (SES) methodology. A significant characteristic of SES, a hierarchical tree structure, enables systems to access network packet information quickly and efficiently. Also, presenting an automated system design is the secondary purpose of this study. Our approach shows adaptive awareness of pragmatic frames (contexts) and makes a network traffic analysis system with high throughput and a fast response time that is ready to respond to user applications. We build models and run simulations to evaluate specific purposes, i.e., analyzing network protocols use, evaluating network throughput, and examining intrusion detection algorithms, based on Discrete Event System

Specification (DEVS) formalism. To speed up evaluation time, we apply a web-based distributed simulation methodology. DEVS/Service Oriented Architecture (DEVS/SOA) facilitates deploying workloads into multi-servers and consequently increasing overall system performance. In addition to the scalability limitations, both tcpdump and Ethereal have a security issue. As well as basic network traffic information such as IP addresses, port numbers, and packet sizes, captured files by these tools contain secure information: user identification numbers and passwords. Therefore, captured files should not be allowed to leak out. However, network analyses need to be performed outside target networks in some cases. The distributed simulation—allocating distributing models inside networks and assigning analyzing models outside networks—also allows analysis of network behaviors out of networks while keeping important information secured.

CHAPTER 1. INTRODUCTION

1.1. Motivation and Goals

During past decades, companies and organizations have constantly been developing computer systems, and as a result, they accumulate huge unorganized, unshared data; they tend to keep the data as simple and local. Therefore, policy makers in companies and organizations obtain needed data by manipulating several processes, or, worst case, they never receive the data that they really want to get. However, decision makers require fast, accurate, and sufficient data in a rapidly changing social environment.

Currently, the network uses, especially the number of internet users, increase rapidly. Also, high quality of service is required, and this requirement often results in sudden network traffic increases. As a result, an efficient system for managing large network traffic datasets becomes an important issue. Network traffic analysis includes the monitoring of all the network behaviors, controlling networks and hosts, and applying network traffic behaviors to achieve an effective management. Network administrators have had difficulties with the lack of consistent traffic analysis or a management system. Lacking a network traffic analysis system with high throughput and fast response time requires that, instead, several processes of manipulating metadata be used, and this results in an insufficiency of useful data for designing network capacity.

There are several network traffic analysis tools such as tcpdump, Ethereal, and other applications. But, these tools are limited. Tcpdump is a powerful tool that allows us to sniff network packets and make some statistical analysis 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 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 analysis. 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 inefficient computational power. Although Ethereal is easier to use than tcpdump, it still limits the size of target-analyzing files. Our experiments show that Ethereal cannot analyze more than two-day network activities in personal computers. To examine more than two-day activities, network managers must control Ethereal by iterating capturing and analyzing processes periodically to avoid excessive system memory uses.

The main objective of this study is to propose an approach to deal with large amounts of information being easily and efficiently shared among organizations. To achieve this goal, we use the System Entity Structure (SES) for the system design. The SES is a theory for designing structured information hierarchically and efficiently, and it is very useful for data engineering. We propose a feasible network traffic analysis system, System Entity Structure based Network analyZER (SES/NZER). We design a behavior which represents network traffic activity and network packet information. The behavior design is based on System Entity Structure (SES) methodology. In addition, we suggest an automated context awareness system for network behavior analyses such as protocols evaluation, network throughput analysis, and intrusion detection evaluation. Every customer may request different analyses. For example, some customers want to evaluate network protocol uses. Other users want to measure network throughput. Depending on various requirements (pragmatic frames), systems need to be optimized for fast and effective analyses. The SES enables systems to be adaptively optimized. This is fundamental automated context awareness. Presenting the automated system design is the second objective of this study. Reactions to users' applications facilitate systems holding accurate data only. Therefore, we could analyze long-term network activities which Ethereal cannot evaluate. The hierarchical tree structure of SES facilitates efficient and fast interpretation of large amounts of data. The two advantages, fast and efficient information sharing and automation, save network maintaining costs and enable rapid reactions against problems such as the necessity of bandwidth increase for protocols/services. 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. DEVS/SOA (DEVS/Service Oriented Architecture) facilitates deploying workloads into multi-servers and consequently increasing overall system performance.

In addition to the scalability problem (the size limitation of capture-files), both tcpdump and Ethereal have security issues. Capture-files, which are evaluated by either tcpdump or Ethereal, include all the information of packets such as IP addresses, protocol types, packet size, and other fundamental attributes. As well as basic network packet information, user IDs and passwords are also contained in captured files. Because captured files hold secure information, Tcpdump and Ethereal are allowed to monitor network behaviors and to capture raw network traffic inside networks with special privilege on some platforms. However, network analyses need to be performed outside target networks in some cases. It 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. Distributed simulation is good solution for analyzing network behaviors remotely while keeping data secured.

1.2. Organization of the Thesis

This study includes background knowledge in chapter 2. Chapter 2 introduces ontology/data engineering methodology for modeling and simulation, System Entity

Structure (SES) theory for representing data engineering, eXtensible Markup Language (XML), and Discrete Event System Specification (DEVS) formalism. In chapter 3, we show relative studies, the Web Ontology Language (OWL), the Unified Modeling Language (UML), and Protégé, and practical examples, a semantic web music service (MusicBrainz), a coding system to classify both products and services (UNSPSC), and a web-based talk system (ITTALKS), with regard to ontology theorem. Chapter 4 illustrates design issues for a network traffic analysis system in details. Intrusion Detection System (IDS) is introduced in chapter 5. We model and simulate a network traffic analysis system (protocols/services evaluation and network throughput analysis) based on DEVS formalism in chapter 6. The experimental results for generic network behavior analysis are also presented in chapter 6. Comparisons between our system and Ethereal are discussed, and we state the problems of our approach in chapter 7. In chapter 8, we present a web-based distributed simulation for a network behavior analysis system and DEVS Service Oriented Architecture (DEVS/SOA). The experimental results of distributed simulation are presented in chapter 9. Lastly, we conclude our study and address future works.

CHAPTER 2. BACKGROUND KNOWLEDGE

There exist modeling approaches for designing and simulating complex systems' specifications. Entity-Relation (ER) [1] is an approach to represent systems' structures such as data entities and their relationships. Other approach is Unified Modeling language (UML), and it is designed to represent objects and their relationships. Even if ER and UML enable modeling systems' specifications and describe various kinds of logical models, each approach is limited in specific modeling. ER is very good for logical model representations but not good enough for visual modeling. On the other hand, UML is good for both logical models and visual models but lacks model persistence [2].

The System Entity Structure (SES) expressed by XML modeling framework is very useful for multi-level (high level applications and low level data specifications) modeling and simulation based on ontology and data engineering methodology. The SES is a hierarchical tree structure with entities and relations. The SES approach shows more potential better for logical, visual, and persistent modeling than either ER or UML.

In this chapter, we introduce background theories such as the System Entity Structure (SES) theory and Discrete Event System Specification (DEVS) formalism.

2.1 Ontology/Data Engineering for Modeling and Simulation

The concept of ontology, which has a long history in philosophy, is a study of reality and the nature of being or existence. In philosophy, ontology (from the Greek, ontos: of being and logos: science, theory) is the study of being or existence and forms the basic subject matter of metaphysics. It seeks to describe the basic categories and relationships of being or existence to define entities and types of entities within its framework. In computer science, a well known definition of an ontology is:

An ontology is a specification of conceptualization [3]

An ontology is a data model that represents a set of concepts within a domain and the relationship between those concepts. It is used to reason about the objects within that domain. Ontologies are used in artificial intelligence, the semantic web, software engineering, and etc. as a form of knowledge representation with respect to the world or some part of it. Ontogolies are commonly encoded using ontology languages such as CycL, KIF (Knowledge Interchange Format), OWL (Web Ontology Language), or any other representation which can define objects, properties and its relations. Ontology languages are formal languages used to construct ontologies. They allow the encoding of knowledge about specific domain and often include reasoning rules that support the processing of that knowledge. T.Gruber [3] stated that knowledge in ontologies can be formalized using five kinds of components: concepts, relations, functions, axioms, and instances.

1. Concepts: calls as classes, collections of abstract, or types of objects.
2. Relation: type of interaction between concepts of the domain
3. Function: mappings between a list of input arguments and its output argument
   - defined as $F: C_1 \times C_2 \times \ldots \times C_{n-1} \rightarrow C_n$
4. Axioms: sentences that are always true
5. Instances: elements of a given concept, actual objects of classes

An Ontology could be clearly conceptualized if these components are well defined. In computer and information science, an ontology is an essential methodology to develop a shared conceptualization in a semantic web. Also, ontology enhances knowledge management methodology by unifying pragmatics of knowledge base system. The huge advantage of ontology is not in processing, but in sharing meaning, emergence and discovery of gaps and for improving a tacit knowledge transfer. Ontology may contain information in a specified declarative language, but it may also include unstructured or unformalized information expressed in a natural language or a procedural code. Computer-based ontology provides formal and structured representation of domain knowledge. It is designed to serve as a raw material for computer reasoning and computer-based agents. The ontology provides a formally defined specification of the meaning of those terms, which are used by agents during their interoperation. Because agents can differ in their understanding of environment, it is important goals capabilities, but they can still interoperate in order to perform a common task.

Data engineering is an aspect of ontology engineering. The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society addresses that "Data Engineering is concerned with the role of data in the design, development, management and utilization of information systems. Issues of interest include database design; knowledge of data and its processing; languages to describe data, define access and manipulate databases; strategies and mechanisms for data access; security and integrity control; and engineering services and distributed systems." [4] The contribution of data engineering is that it enables easy and efficient information sharing among services in organizations. Also, data engineering makes meaningful information be exchanged, not just simply bits and byte data. Model and Simulation-based data engineering relies on rigorous principles to ensure that metadata structures can be designed, represented, constructed and implemented in such a way that facilitates automated comparison and analysis, translation, and implementation [5]. In this study, model and simulation-based data engineering is applied to support flexibly applicable data exchange using data models.

Recall that ontology describes a state of the world and its changes over time. Information exchange occurs when a person or information system reports a new state of the world, or changes in previous states, to another person or information system. The person or system that generates and reports the information is called a *producer* while the user is called a *consumer*. The principle that this document emphasize is that the consumer's *use* of the information should determine the description mechanism, or ontology, used by the producer. We'll formulate a means of characterizing the consumer’s use of the information and call it a *pragmatic frame*. The developer of the ontology, also called the data engineer, has the task of tuning the ontology to the pragmatic frame. Context awareness by ontology/data engineering could be developed in modeling and simulation approach. For more understanding, an example of social relations on an island is presented.

2.1.1 Ontology for Social Relations on an Island

There are four people stuck on an island: *a*, *b*, *c*, and *d* (you are free to associate your favorite movie or sitcom characters with these letters). Because people in such a situation frequently change their affection for each other, we want to represent in some way their daily likes and dislikes of one another. We begin with the following matrices

Likes

	a	b	c	d
a		1	1	
b	1			
c	1			1
d			1	

Dislikes

	a	b	c	d
a				1
b				
c		1		1
d	1			

Using this framework, there is a large but easily computable number of possible states of affection that characterize this small number of people. Indeed, there are 4 × 4 = 16 cells in the *Likes* matrix, each of which can independently have a 1 or a blank. Therefore, there are 216 states of the *Likes* matrix and, again assuming independence, the same number for the *Dislikes* matrix, for a total of 232 that is the product of the two groups. In reality, the number of states is much smaller because the following constraints:

1. No one likes or dislikes himself or herself;
2. Affection is mutual and so is disaffection (for example, you like someone who likes you, and the same applies to disliking someone);
3. Liking and disliking are mutually exclusive — you can't like and dislike someone at the same time;
4. To quote an ancient proverb, "The enemy of my enemy is my friend."

2.1.2 Definition of Ontology

Numerous definitions of the concept of ontology can be found in several literatures. We'll employ one that is particularly appropriate to the point of view taken in this study. The Federation of Intelligence Physical Agents (FIPA) provides a discussion of ontology that can be found on the web [6]. FIPA's concept of an ontology is a logical formalism that tries to express a set of possible world structures through its family of models. The definition itself tries to make clear that the language with its axioms usually cannot exactly capture the set of world structures that the definer intends, so it is an approximation to this intended conceptualization. In our approach, we will not necessarily restrict the defining mechanism to a logical formalism. For example, eXtensible Markup Language (XML) [7] might be used to capture a set of possible world states while itself not being explicitly formulated within the approach of classical logic. We will explicitly incorporate a concept of *pragmatic frame*, which delineates a data engineer's domain of interest and relates an ontology as being adequate or not to this domain. It will turn out, for example, that without further validity-checking tools, XML cannot represent the worlds in which enemies of enemies are always friends. Further, we will also want to explicitly consider how world states can change over time. For example, it might be that our islanders can't get along with each other for more than one day at a time, therefore, if *x* likes *y* today, then *x* will dislike *y* tomorrow. Further, suppose that it takes a whole week for people who have a falling out to reconcile. So, if *x* dislikes *y* today, then only this time next week will the corresponding "1" in the *Dislike* matrix disappear. Thus, we'll be interested not only in static data engineering but also in dynamic data engineering, and this will lead us toward including the full capability of modeling and simulation.

2.1.3 Pragmatics: The Information Exchange Framework

A producer observes the state of the world at its location and encapsulates it in a message to a consumer that stores it for later use. As we noted previously, the world state is a member of a set delineated by an ontology that reflects the designer's conceptualization. But what could that conceptualization be? By taking into account the *pragmatics* of the situation, we get a way to answer this question. By pragmatics we mean the aspect of language that has to do with how the information transmitted by a message will be used. In Figure 1, we can point out how the stored data will be used later.

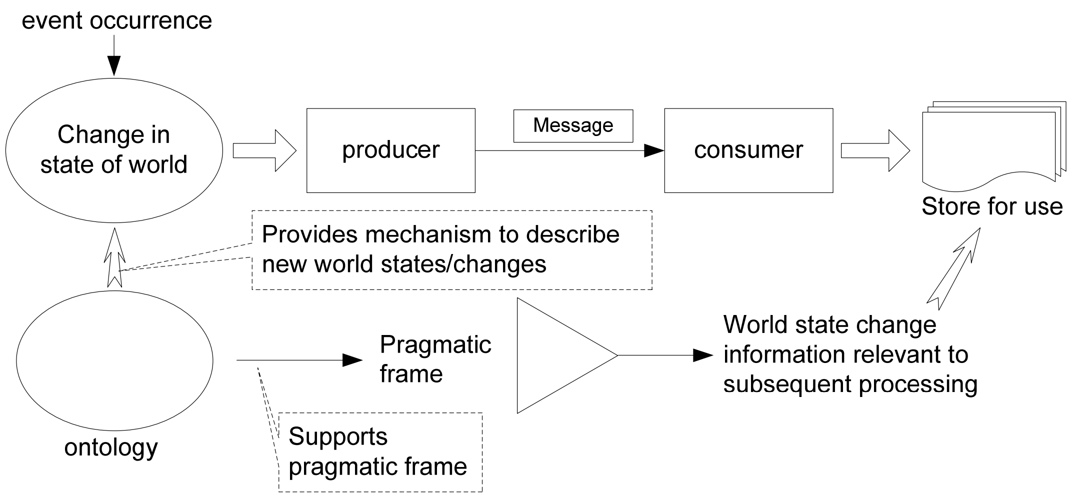


Figure 1. Information exchange framework [5]

Consider two examples

1. *Information Sent by a Dealer to Department of Motor Vehicles*

   Whenever you buy a new car, some information is sent from the dealer to the Department of Motor Vehicles where you live, which stores it in its database. The pragmatics of the use here is the subsequent processing of this data — it will be used when you register the vehicle and pick up your plates, and to compute taxes and annual renewals.

2. *Information Sent by a Dealer to Manufacturer's Headquarters*

   Every month a car dealership sends a sales report to the manufacturer's headquarters to allow it to assess inventory levels and plan its production schedule for the next month. Table 1 illustrates the framework.

Event causing world state change	**Producer**	**Consumer**	**Pragmatics: Subsequent use**	**Message Contents**
New car purchase car ownership transferred from dealer to buyer	Dealer	Department of Motor Vehicles	Register owner's vehicle and give out plates	Buyer and vehicle identification
New car purchase car ownership transferred from dealer to buyer	Dealer	Manufacturer Headquarters	Assess inventory levels and production schedule	Number of cars of each make and model that were sold during the month

Table 1. The purchase of the car caused a world state change

In our framework, an event occurred when you bought the car. This changed the world because the car left the dealer's lot and it is now yours to drive. The producer is the same — the car dealer — but the pragmatics are different in the examples just given: the consumers are different (Department of Motor Vehicles and manufacturer) and their subsequent use of the received information will be different. The point is this: notice how the pragmatics determines the nature of the data to be sent, namely, the contents of the message. In the first case, the dealer needs to provide specific information about the buyer (name, address, etc.) and vehicle (identification number, make, model, etc.). In the second case, this information is not relevant. Instead, the dealer needs to inform the manufacturer of how many cars of each make and model he sold during the month.

We'll say that an ontology *supports* (or *is applicable to*) *a pragmatic frame* if the world states (or state changes) that it can describe include those that are needed by the frame. In other words, the message contents encode the right information for the intended use. Notice that an ontology can be designed to describe a set of world states for some domain. This might be feasible for a limited situation such as the states of affection of four people on an island. However, in many situations, it would be much more efficient to describe the change in state caused by an event in contrast to the new state that was engendered. So we allow for ontologies that describe state changes rather than states. For example, in the case of an island with one million people, we might explicitly convey the few pairs of people that started, or ceased, liking each other since the last update. Indeed, we'll later discuss such change-based updating in depth. For the moment, we note that the producer and consumer must have agreed initially that entries in the database remain valid unless explicitly updated.

An ontology is *minimal* for a frame if it supports only that frame, not a larger one. For example, we could have minimal ontologies for the vehicle registration and inventory update frames, respectively. As illustrated in Figure 2, the first ontology would provide a means to describe the buyer's name, address, and Social Security number and the vehicle's make, model, and identification number. The second would provide a means to describe the number of cars of each make and model sold during the last month. On the other hand, we could have one larger, more general ontology to cover both needs (and possibly others). The minimalist approach offers conciseness and efficiency, whereas the generalist approach potentially offers better integration and, if designed with forethought, potential extensibility to meet future demands. There will always be a tradeoff between these approaches, but the *scope* of the ontology — the part of the world that it can help to describe — must ultimately be limited by a deliberate choice of the developer.

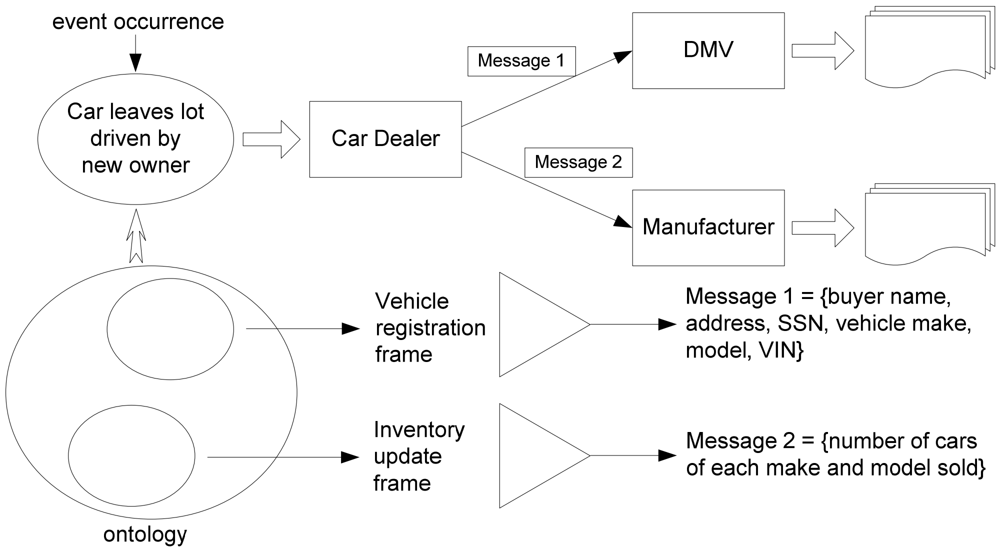

event occurrence
Car leaves lot driven by new owner
Car Dealer
Message 1
Message 2
DMV
Manufacturer
ontology
Vehicle registration frame
Inventory update frame
Message 1 = {buyer name, address, SSN, vehicle make, model, VIN}
Message 2 = {number of cars of each make and model sold}

Figure 2. Car purchase example of the information exchange framework [5]

2.1.4 Ontology/Data Engineering based Modeling and Simulation

Two levels, those of ontology and implementation, constitute the overall architecture for our simulation-based ontology engineering methodology. As depicted in Figure 3, at the ontology level, the modeler creates an ontology to satisfy the pragmatic frames of interest in a given application domain. The pragmatic frames can be specified in XML, via restricted natural language, through a GUI which is the SESBuilder [8]. It is then automatically encoded to an XML schema/document type definition (XSD or DTD) at the implementation level. Such automation is an important advantage, since other ontology developments based on UML currently lack the combination of automation and a broad range of tools that the System Entity Structure (SES) framework supports. The XML instance documents specified by a schema are formally represented by the family of pruned entity structures (PES) at the ontology level. In the context of dynamic data engineering, each completely pruned PES specifies a Discrete Event Simulation (DEVS) simulation model that constitutes a capability to describe world states that evolve in time. When limited to static data engineering, the family of PES represents a logically possible set of world states. Each such state is like a snapshot of the world as depicted by the ontology in service of the application contexts (the pragmatic frames). An XML document instance is the concrete encoding of the abstract PES. It encodes data in an information exchange that either directly represents a world state in the static case, or can be transformed to a simulation model in the dynamic case. Finally, those of ontology can be modeled and simulated.

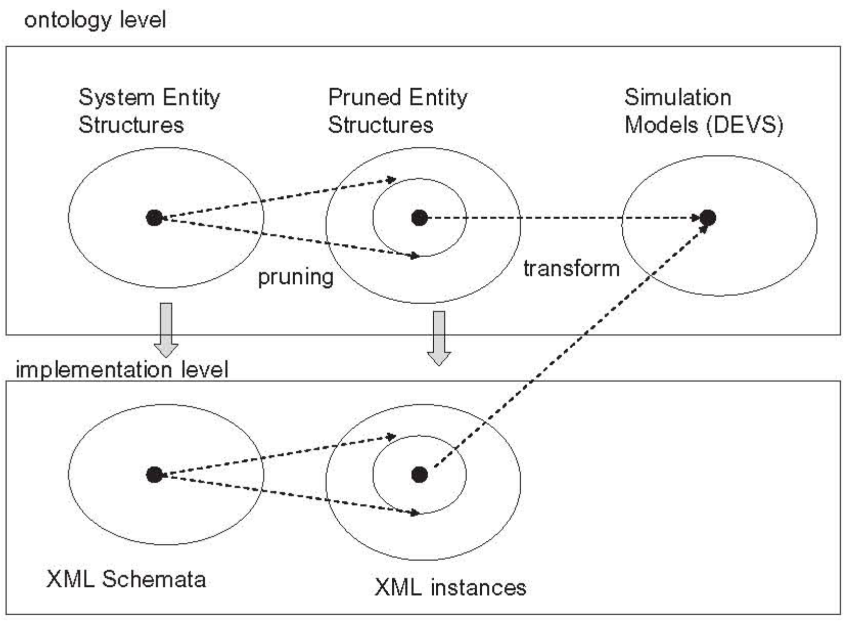


Figure 3. Simulation-based data engineering methodology [5]

2.2 System Entity Structure (SES)

The basic concept of the System Entity Structure is that a system entity represents the real system enclosed within a certain choice of system boundary. In a real system, many system entities and the experimental frames are dealt. 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. Professor Zeigler proposed the System Entity Structure (SES) [9, 10] and the SES is a theory to design systems hierarchically and structurally. The basic idea of the SES is that a system entity represents the real system enclosed within a certain choice of system boundary. In real system, many system entities and the experimental frames are dealt. Thus it is necessary to organize the model and experimental frames around the structure. The SES includes entities and their relationships. Table 2 presents the key components consisting of the SES.

Components	**Descriptions**
Entity	A real world object which either can be independently identified or is postulated as a component in some decomposition or a real world object. e.g., building
Entity-Aspect (|)	A decomposition, a way to break down an entity into parts or components (entities). The children of an aspect are entities representing components in a decomposition of its parents. e.g., door, roof, and etc.

Multiple Entity (|||)	A multi-decomposition, relationship between multi entities and an entity. e.g. doors and front door and back door.
Entity-Specialization (||)	A classification, a way to classify an entity into special cases or subclasses. The children of a specialization are entities representing variants of its parent. e.g., home, office, store, or etc.

Table 2. Components of System Entity Structure (SES)

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 [9]. The pruned entity structure can be transformed into a composition tree and eventually synthesized into a simulation model. 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 DTD or Schema. Entity, Aspect, Multi-Aspect, and Specialization build the primary components in DTD or Schema. At the ontology level, the modeler develops one or more SESs depending on models, and the SESs are merged to create an ontology in order to satisfy the pragmatic frames of interest in a given application domain. An SES can be specified in various ways, and then it is transformed to an XML schema or an XML document type definition (XSD or DTD) at an implementation level. The pruning operation of SESs creates pruned entity structures (PESs), and the PESs transform to simulation models.

In chapter 2.3, we provide an overview of a language, which is eXtensible Markup Language (XML).

2.3 Extensible Markup Language (XML)

The Extensible Markup Language (XML) is a W3C-recommended general-purpose markup language for creating special-purpose markup languages, capable of describing many different kinds of data [7]. XML is an extensible language not like HTML. HTML limits tags within certain syntax. Otherwise, users could define tags according to contents of documents and let others use the defined tags in XML. XML is a language for expressing other languages, that is a meta language. As a result, XML will be widely used due to its characteristics of platform independency. Also, since XML substitutes all kinds of data structure and supports various data format [11], it extends its uses to search engines, data engineering, and etc.

2.4 Discrete Event Simulation

2.4.1. Fundamentals of Computer Simulation

Computer simulation is an activity of representing the temporal behavior of a physical or a conceptual system for a specific period of time. A simulation model is a specification representing the system in terms of a set of states, events, and behavior functions. Simulation time can be slower, faster, or equal to physical time. Also, time resolution can be arbitrarily defined [12].

During simulation, the current status of a model is represented by a state. A state transition occurs just before initiating or after completing a particular behavior. A state feasibility test may be involved before a state transition happens. An event is a data object that is produced and consumed by simulation components: e.g., logical simulator and coordinator. If necessary, a set of events is exchanged among those components in order to complete a simulation task. A behavior function is invoked when events are received or produced by a model or a specific behavior of the model is to be performed.

Simulation is classified into continuous simulation and discrete simulation according to the state transition occurrence interval. During a simulation, if a state transition occurs continuously in time, the simulation is a continuous simulation. While, if state transitions happens in discrete time, the simulation is called a discrete simulation. In a discrete simulation, if state transitions occur in term of discrete time interval (or time steps), the simulation is referred to as a time driven discrete simulation (or discrete-time driven simulation). An event-driven discrete simulation (or discrete-event driven simulation) is defined if state transitions happen based on event activities. Figure 4 depicts the classification of a computer simulation.

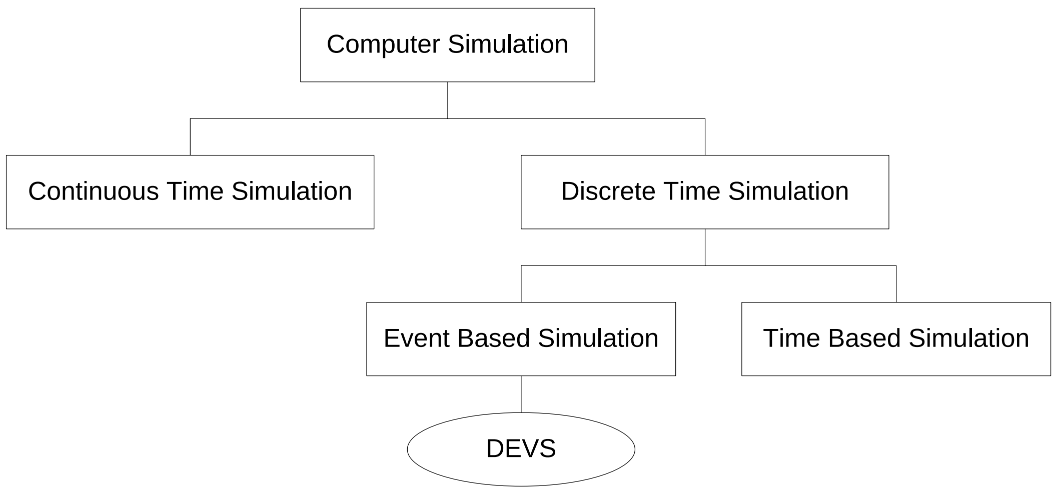


Figure 4. Classification of computer simulation

Depending on the simulation time synchronization scheme, a simulation is viewed as a conservative or an optimistic activity at a specific time. All simulation activities are completed before advancing time and time must be synchronized in the conservative scheme [13, 14]. In the optimistic scheme, time does not need to be synchronized globally and simulation activities at a particular time need not all be completed before advancing time. Only when a time causality problem occurs, time needs to be synchronized [15, 16]. Conservative schemes guarantee all activities are performed without any time causality problems. By loosening the time causality constraint, optimistic schemes perform better for certain simulation problems that contain a high degree of parallelism between simulation models. However, it requires additional memory to keep information in regards to activities that occurred at a previous time. When time causality problems happen, current simulation time rolls back to a previous time that did not violate time causality. Generally, the performance of the simulation is not directly associated with a simulation time synchronization scheme but instead is related to the nature of the given simulation problem [12].

2.4.2. Discrete Event System Specification (DEVS)

The Discrete Event System Specification (DEVS) is a formalism providing a means of specifying a mathematical object called a system [17]. 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 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 models or coupled models. 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., external transition function (δ_{ext}), internal transition function (δ_{int}), output function (λ), and time advance function (ta())) to describe the dynamic behavior of the model. Figure 5 illustrates the system representation of an atomic model.

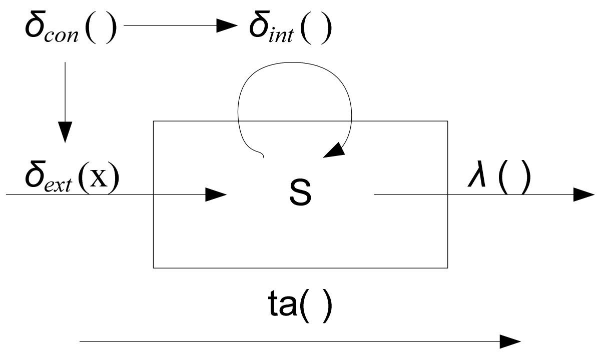


Figure 5. System representation of atomic model [17]

The external transition function (δ_{ext}) carries the input and changes the system states. The internal transition function(δ_{int}) changes 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()) function adjusts simulation time after generating an output event. The Atomic model is specified as follows:

Atomic model:

$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$

where,

X: set of external input events;

S: set of sequential states;

Y: set of outputs;

$\delta_{\mathrm{int}} : S -> S$: internal transition function

$\delta_{ext} : Q \times X^b -> S$: external transition function

where,

Q = {(s, e)| s$\in$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,

$\lambda : S -> Y^b$: output function generating external events at the output

$ta : S -> R^+_{0,\infty}$: time advance function;

Basic models may be joined in the DEVS formalism to form a coupled model. A coupled model is the major class which embodies the hierarchical model composition constructs of the DEVS formalism [17]. A coupled model is made up of component models, and the coupling relations which 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 which create the desired communication networks. A coupled model is defined as follows and a coupled model can be seen in Figure 6:

Coupled Model:

$DN = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} >$

where,

X: set of external input events;

Y: a set of outputs;

D: a set of components names;

for each i in D,

M_i is a component model

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 [18]:

- 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
  - Internal coupling (IC) connects output ports of components to input ports of other components

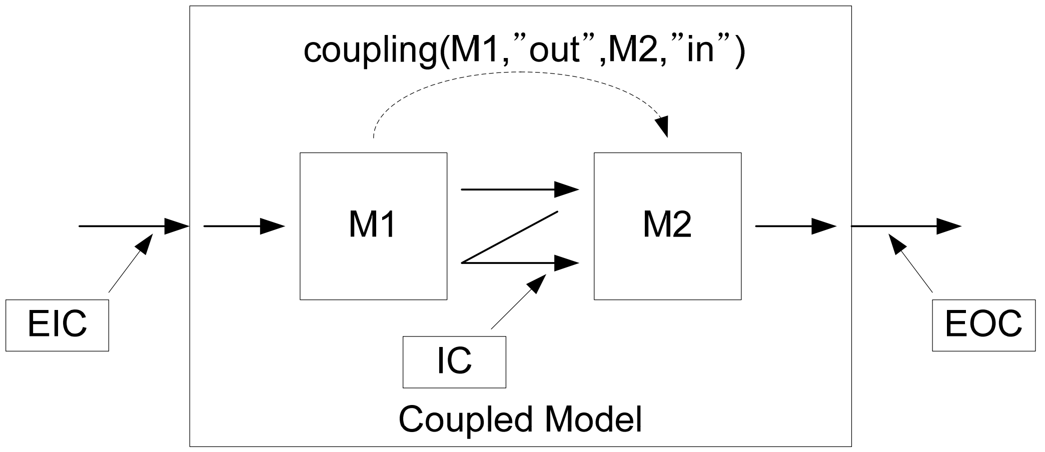


Figure 6. An example of coupled model [17]

2.4.3 Experimental Frame

An experimental frame is a specification of the conditions under which the system is observed or experimented with [17]. As such, an experimental frame is the operational formulation of the objectives that motivates a modeling and simulation object. For example, out of the multitude of variables that relate to a forest, the set {lightning, rain, wind, smoke} represents one particular choice. Such an experimental frame is motivated by the interest in modeling the way lightening ignites a forest fire. A more refined experimental frame would add the moisture content of the vegetation and the amount of unburned material as variable. Thus, many experimental frames can be formulated for the same system and the same experimental frame may apply to many systems. Basically an experimental frame specifies a limited set of circumstances under which a system (real system or model) is to be observed or subjected to experimentation. The objectives and experimentation play a role in the modeling enterprise at least equal in significance to model construction. However, in current modeling and simulation world, the statement of objective is not formalized and cannot play its proper role in a computer supported methodology. Experimental Frame demonstrates how the statement of objectives can be operationalized in a process whose product is the formulation of experimental frames. Initial objectives lead to asking specific questions about the real system which in turn require that suitable variables be defined. Such a choice of variables is represented in experimental frames which also express constraints on the trajectories of these variables.

There are two equally valid news of an experimental frame. One views a frame as a definition of the type of data elements that will go into the data base. The second views a frame as a system that interacts with the system of interest to obtain the data of interest under specified condition. In this view, the frame is characterized by its implementation as a measurement system or observer. In the implementation, a frame typically has three types of components: generator, acceptor, and transducer. Generator generates input segments to the system. Acceptor monitors an experiment to see the desired experimental conditions are met. Transducer observes and analyzes the system output segments. Figure 7 illustrate experimental frame and its components.

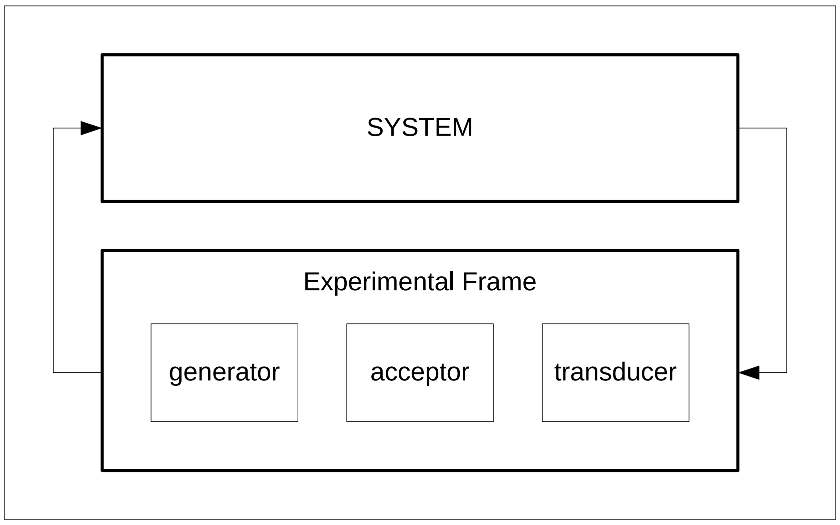


Figure 7. Experimental frame and its components

■ ***Component 1 : Generator***

Generator is an active DEVS which is input free and stimulates the system with input trajectories. Generator is capable of scheduling itself for a state transition, producing an output as a function of this state, and rescheduling itself for the next such iteration. The output function is defined in such a way that an output segment produced by such a generator when started in an initial state satisfies the constraint of a DEVS segment. Generators may be used to implement arrival processes among other classes of input segment to models. In continuous systems such trajectories take the form of various kinds of periodic functions such as steps and ramps, or various periodic functions such as sine waves or square waves. In discrete event systems, arrival of events may be periodic or stochastic. In the latter case, we need to specify the probability distribution of the inter arrival times, such as uniform or exponential. The type of processing required is also part of the generator specification.

- ***Component 2 : Acceptor***

We often make a distinction between the transient and steady state characteristics of system behavior. The acceptor is the slot in the experimental frame where conditions limiting the observation of behavior, such as steady state versus transient, can be specified. An acceptor is a passive DEVS with state set partitioned into two sets, the accepting and non accepting states. In addition the acceptor designates a state in which the system is always initialized. An input segment is accepted by such a system, if it causes the acceptor to reach an accepting state at the end of its application.

- ***Component 3 : Transducer***

The transducer processes the output trajectories, where such post processing may range from none at all to very coarse summaries where only certain features of interest are extracted. In discrete event systems, we might be interested in the turnaround times required to process jobs or in the throughput, rate of job completion. Utilization of various resources and of special events such as failure or blocking may be interest.

Usually the many numbers produces are summarized into statistical quantities such as the average, maximum, or minimum. In experimentation, input variables are those variables that will be treated as influencing the system under study, and output variables are those capable of direct measurement and mediating the computation of variables of interest for the modeling objectives. There is no output segment specification since these are determined by the system during experimentation. When a computation involved in computing interest variables from mediating variables is moved to the experimental frame stage, these computations are expressed as the summary mapping of the frame. The appropriate concrete form of specifying such mapping is the transducer.

2.5 Web Service

A web service [19] is a software system for communicating between a client and a server over a network with XML messages called Simple Object Access Protocol (SOAP) [7, 20]. The web service makes the request of machine-to-machine or application-to-application communication possible with neutral message passing even though each machine or application is not same domain. Such interoperability among heterogeneous applications is realized by web service providing a standard means of communication and a platform independency.

Web services technologies architecture [21] is based on exchanging messages, describing web services, and publishing and discovering web service descriptions. The messages are exchanged by SOAP messages conveyed by internet protocols. Web services are described by Web Services Description Language (WSDL) [22] which is

XML based language providing required information, such as message types, signatures of services, and a location of services, for clients to consume the services. Publishing and discovering web service descriptions is managed by Universal Description Discover and Integration (UDDI) [23] 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 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 8 illustrates the basic Web services architecture describing three roles and operations with WSDL and SOAP.

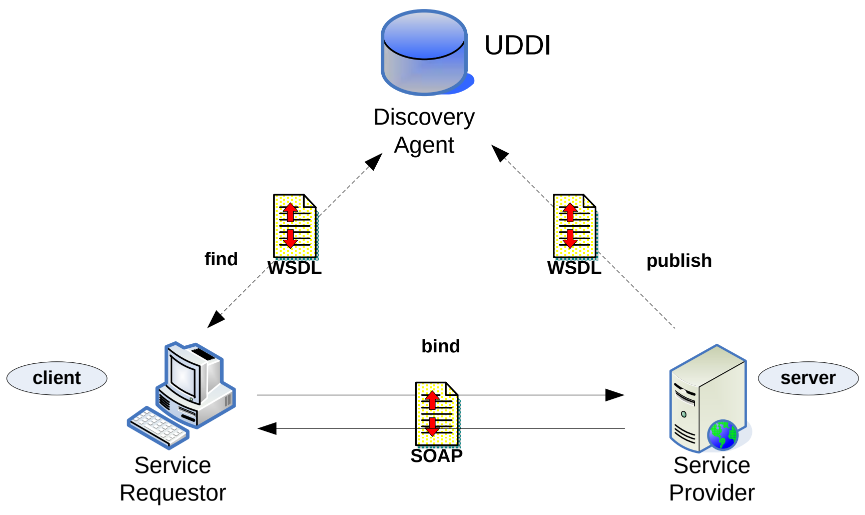


Figure 8. Web services architecture

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

Web services are invoked by many ways but the common use of web services is categorized to three methods such as Remote Procedure Call (RPC), Service Oriented Architecture (SOA) [24], and Representational State Transfer (REST) [25]. RPC Web services was the first web services approach which had a distributed function call interface described in the WSDL operation. Though it is widely used and upheld, it does not support loosely coupled concept for reasons of mapping services directly to language-specific functions calls. A web service is an implementation of Service-Oriented Architecture (SOA) concepts, which means a message is important unit of communication regarded as "message-oriented" services. This approach supports a loose coupling concept focusing on the contents of WSDL. REST Web services focuses on the existence of resources rather than messages or operations. It considers WSDL as a description of SOAP messaging over HTTP, or is implemented as an abstraction on top of SOAP.

CHAPTER 3. STATE OF THE ART

In this chapter, we illustrate relative studies such as the Web Ontology Language (OWL), the Unified Modeling Language (UML), and Protégé. Also, we introduce practical ontology engineering examples such as a semantic web music service (MusicBrainz), a coding system to classify both products and services (UNSPSC), and a web-based talk system (ITTALKS).

3.1 The Web Ontology Language (OWL)

The Web Ontology Language (OWL) [26] is a W3C recommended ontology standard under development to support such intelligent queries. OWL is designed not only for understanding human-readable presentation of content but also for developing applications for information processing. OWL's wide vocabulary and formal semantics enhance machine-interpretability of web contents comparing to Extensible Markup Language (XML), Resource Description Framework (RDF) [27], or Resource Description Framework Schema (RDF-S) [28]. The OWL language provides three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full. The standard example to explain the goals and capabilities of OWL is the wine agent who must be is to be capable of searching the web for answers to queries such as what wine would go with a particular dinner course. The basic ontology framework employs classes, class hierarchies, and properties, which are standalone binary relations. The primary objective of OWL is to support automated logical reasoners that can derive new inferences when applied to combined ontologies from multiple web sites. OWL is intended to operate in the open web environment where ontologies are not developed under central control. Since information cannot be retracted, reasoners must be prepared to cope with contradictions that can arise from merging diverse independently developed ontologies. This represents a departure from earlier applications of formal logic that employed brief belief revision mechanisms to maintain logical consistency under the addition of new knowledge. A research project at Stanford University, Protégé, is a free, open source ontology editor and knowledge-base framework which is implemented for the Open Knowledge Base Connectivity (OKBC) [29] compatible knowledge model and OWL.

3.2 The Unified Modeling Language (UML)

The Unified Modeling Language [30] is the Object Management Group’s [31] most used specification, and the world models not only application structure, behavior, and architecture, but also business process and data structure. The Object Management Group’s (OMG) Unified Modeling Language (UML) helps to specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of these requirements. UML can be used for business modeling and modeling of other non-software systems too. Using any of the large number of UML-based tools on the market, future application requirements and design of a solution that meets them can be analyzed, representing the results using UML's standard diagram types. Recently, OMG have improved with a major update, UML 2.0. UML 2.0 defines thirteen types of diagrams. Figure 8 helps to understand the diagrams by a hierarchically categorized tree structure.

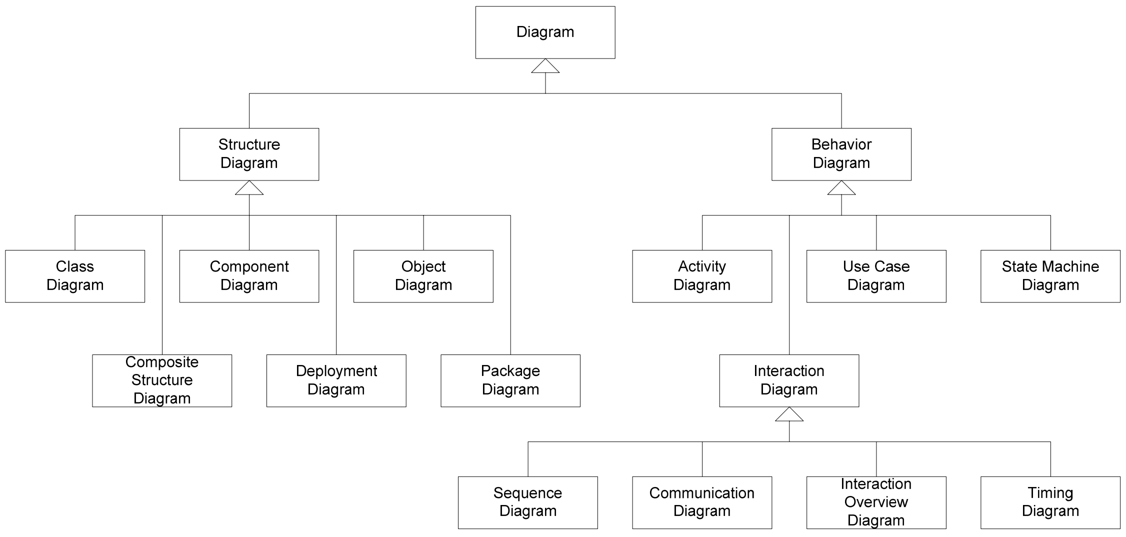


Figure 9. UML diagrams: hierarchically categorized [32]

1. Structure diagrams: Things what should be in systems
   - Class Diagram
   - Component Diagram
   - Composite Structure Diagram
   - Deployment Diagram
   - Object Diagram
   - Package Diagram

2. Behavior diagrams: Things that happen in systems

- Use Case Diagram
- State Machine Diagram
- Activity Diagram

3. Interaction Diagrams: Flows of controls and data in systems
    - Sequence Diagram
    - Communication Diagram
    - Interaction Overview Diagram
    - UML Timing Diagram

3.3 Protégé

Protégé is an open-source development environment for ontologies and knowledge-based systems. It is a tool supporting the construction of ontologies, and it also provides an application platform for knowledge-based systems and libraries for application building. Protégé was developed at Stanford University. It is the best-known ontology editor with plug-ins that supports OWL and enables the following [33]:

- loading and saving OWL and RDF ontologies,
- editing and visualizing OWL classes and their properties,
- defining logical class characteristics as OWL expressions,
- executing reasons such as description logic classifiers,

- editing OWL individuals for Semantic Web markup.

Protégé has flexible architecture and is easy to configure and extend. Protégé has an open-source Java API for the development of custom-tailored user interface components or arbitrary semantic web services. There are several other ontology editors such as OilEd, OntoEdit or DUET. However, Protégé, with its plug-in architecture, gives much wider possibilities. Powerful associations, which consist of developers, universities, governments, and organizations, have supported Protégé, and continuous updates and revisions are promising strengths of Protégé. Protégé-2000 was first published in 1999, and Protégé 4.0 alpha is available now, December 2007. Currently, there are 82,980 registered users. The Protégé platform supports two main ways of modeling ontologies via the Protégé-Frames editor [34] and Protégé-OWL editor [35].

The Protégé-Frames editor enables users to build and populate ontologies that are frame-based, in accordance with the OKBC. In this model, an ontology consists of a set of classes organized in a subsumption hierarchy to represent a domain's salient concepts, a set of slots associated with classes to describe their properties and relationships, and a set of instances of those classes—individual exemplars of the concepts that hold specific values for their properties. The Protégé-OWL editor enables users to build ontologies for the Semantic Web, in particular in the W3C's Web Ontology Language (OWL). An OWL ontology may include descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined with using defined OWL “mechanisms.” In addition, Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema.

3.4 Practical Ontology Examples

In this chapter, we introduce three practical examples for ontology engineering. The first example is MusicBrainz which is a semantic web music service. The second example is UNSPSC, which stands for United Nations Standard Products and Services Code, a coding system to classify both products and services. The last one is a web portal offering access to information about talks, seminars, and colloquia related to information technology, ITTALKS.

3.4.1 MusicBrainz: A Semantic Web Music Service

MusicBrainz is a music metadatabase, and its purpose is to create a comprehensive open-content music database. It systematically manages music metadata, which is created by various organizations, through an ontology. MusicBrainz includes a large database of music metadata. It contains information about 349,693 artists, 533,749 albums, and 6,275,935 tracks at Dec, 2007 [36]. The MusicBrainsz metadata provides data about music, such as artists, album titles, and tracks, but not the music itself.

MusicBrainz is one of the first the Semantic Web Services [37]. It combines the principles idea of the Semantic Web and web services together. The Semantic Web is a project to create machine-processable information into human-readable contents on the web. The web service is very similar to the Semantic Web, but it aims to enable interaction with machine-interpretable information over a network. It assigns a Uniform Resource Identifier (URI) about artists, albums, tracks, and other such information to its database, and it uses RDF to address information. Instances are expressed only by URIs and can escape metadata redundancy and reduce error rates. In this system, an ontology facilitates mapping distributed metadata.

3.4.2 UNSPSC: Coding System to Classify both Products and Services

The United Nations Standard Products and Services Code (UNSPSC) provides an open, global multi-sector standard for efficient, accurate classification of products and services [38]. It is a standard to classify hierarchically all the products and services throughout the global eCommerce marketplace and eBusiness. The UNSPSC services a single global classification system that can be used for the following:

- Company-wide visibility of spend analysis
- Cost-effective procurement optimization
- Full exploitation of electronic commerce capabilities

An ontology is used for standardizing common vocabulary items about products, and the ontology links together scattered products found on the web. The common vocabulary items of the ontology facilitate information exchanges among the products on the web.

3.4.3 ITTALKS: web-based talk system

ITTALKS is a web portal offering access to information about talks, seminars, and colloquia related to information technology [39]. It is a DARPA Agent arkup Language (DAML) [40, 41] based system that enables both users and agents interaction. ITTALKS implements ontologies to describe talks and details about the talk, such as people, locations, and discussion topics. The ontologies are connected in ITTALKS system, but they are independent. In this independently cooperative environment, each ontology develops by itself, and individual advances promote the complete ITTALKS system.

CHAPTER 4. DESIGN ISSUES OF SES/NZER

The goals of a network traffic analyzer are to help network administrators to manage very complicated network topology and to increase efficiency for secure and effective data transfer. The network use, especially the number of internet users, increases rapidly. Also, high quality of service is required, and this requirement results in sudden network traffic increases. As a result, designing efficient systems for managing large network traffic data becomes an important issue. Ontology/data engineering methodology is used to build an effective system for analyzing large amounts of network traffic data. The primary objective of this study is to develop a system that allows easy and efficient information sharing among organizations. The SES and XML modeling approaches allow systems to easily handle huge amounts of data, and the two approaches facilitate the modeling and simulation 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. This study illustrates how to analyze network behaviors that are requested by customers. Both protocol analysis and network throughput analysis are provided as pragmatic frames (users' applications). Therefore, we design a behavior (SES) reflecting network packet transmissions and packet information. The SES includes a methodology for interpreting metadata in SES/NZER. This chapter shows the overview of developing processes, i.e., the way of capturing network traffic data and analyzing the data, of SES/NZER. Figure 10 shows the developing processes of

SES/NZER. We classify three levels: *Design*, *Implement*, and *Analyze* levels. In the *Design* level, we design an SES for network traffic behavior and capture network traffic in a subnet. In the *Implement* level, the SESBuilder [8] generates an XML schema for an SES. From an XML schema, XML instance files including data attribute values that are created through pruning operations at the *Analyze* level. Also, we run a simulation (DEVS modeling and simulation) and evaluate results. The simulation framework is automatically re-structured according to customers' requirements (pragmatic frames). Developing an automated system design is the secondary goal in this study.

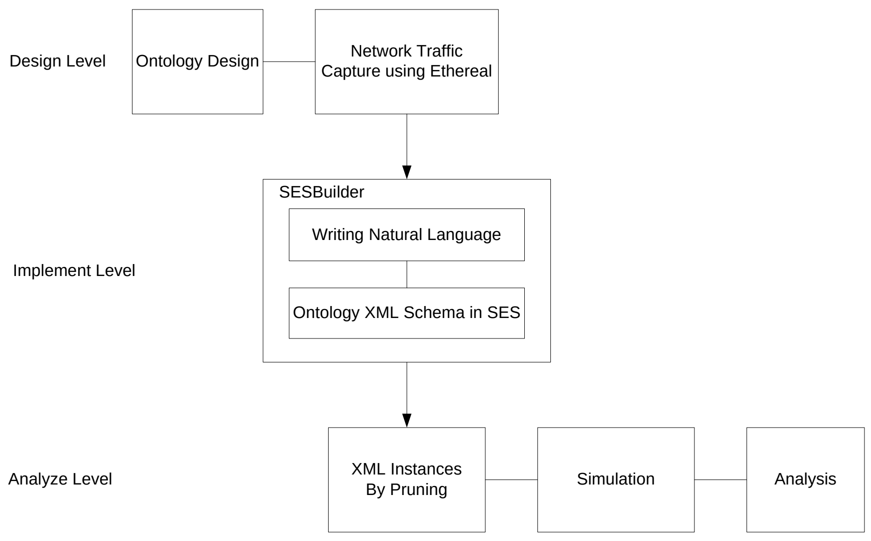


Figure 10. Developing process of the network traffic analysis system

4.1 SES design for network traffic analysis

In this chapter, 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 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 network traffic analysis. 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 10 is a hierarchical SES tree structure representing network packet behaviors.

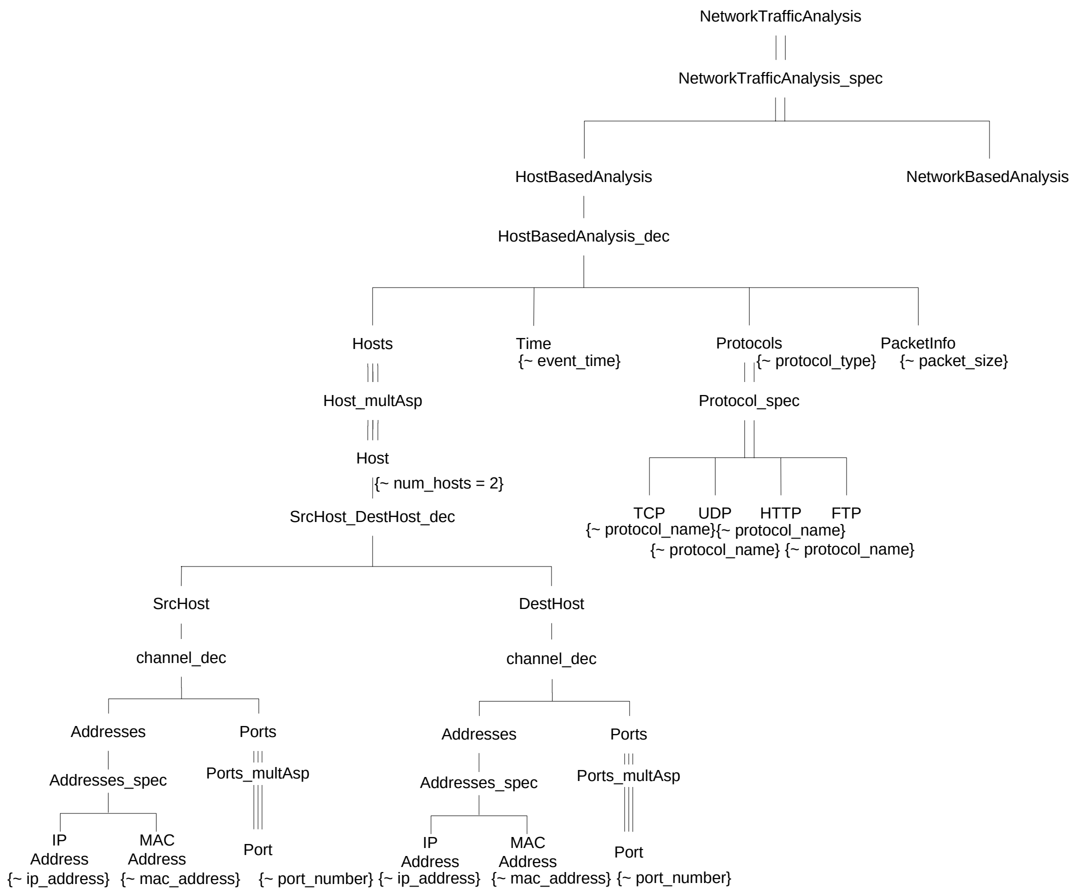


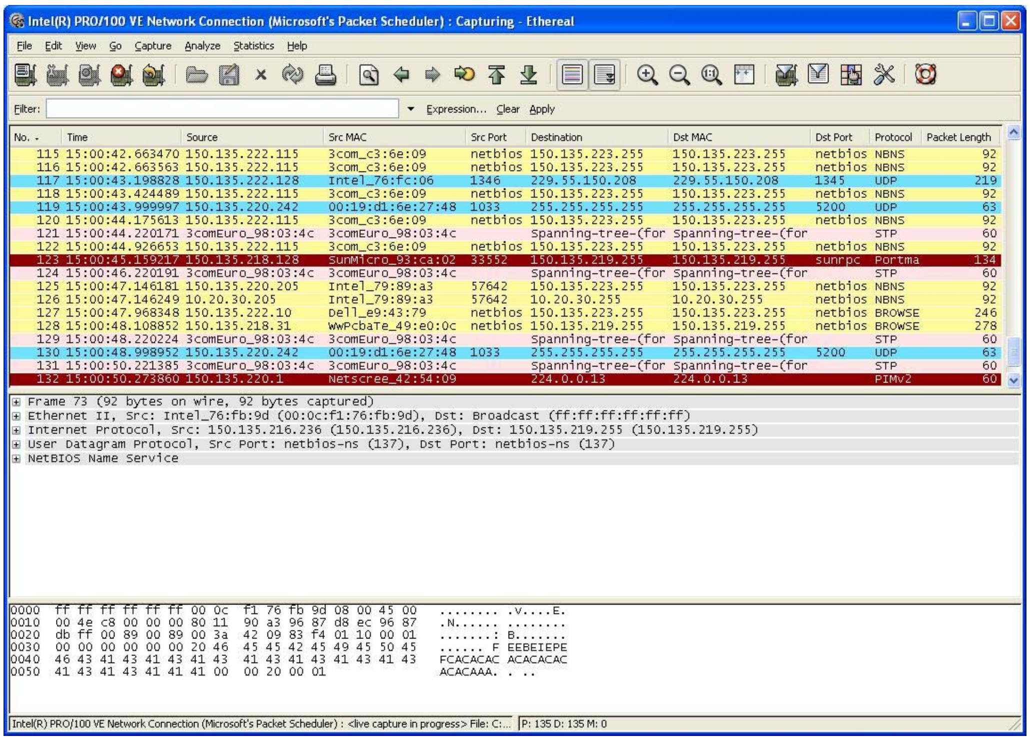
Figure 11. System Entity Structure (SES) of network traffic behavior

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* has the *protocol_type* attribute. The *Protocols* can be implemented with the *TCP*, the *UDP*, the *HTTP*, or the *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.

Based on this SES, we monitor network activities and capture the fundamental packet information which is mentioned in this chapter. We also design simulation models for network traffic analysis, and run simulations to examine how efficient the SES based data engineering methodology is in the research areas of network protocols/services and throughput evaluations.

4.2 Network Traffic Data

To evaluate network traffic behaviors, network traffic data is required. Network traffic data could be real data, virtual data (simulated experimental results), or existing dataset such as KDD 1999 Cup dataset [42]. In this study, we use real network traffic data. To obtain real data, we monitor network behaviors in a subnet of Arizona Center for Integrative Modeling and Simulation (ACIMS) lab [43] in the department of electrical and computer engineering at the University of Arizona. We use the Ethereal [44], which is a well-known network protocol analyzer, for capturing network behaviors. Figure 11 shows a screen shot of the Ethereal.


Intel(R) PRO/100 VE Network Connection (Microsoft's Packet Scheduler) : Capturing - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

No.	Time	Source	Src MAC	Src Port	Destination	Dst MAC	Dst Port	Protocol	Packet Length
115	15:00:42.663470	150.135.222.115	3com_c3:6e:09	netbios	150.135.223.255	150.135.223.255	netbios	NBNS	92
116	15:00:42.663563	150.135.222.115	3com_c3:6e:09	netbios	150.135.223.255	150.135.223.255	netbios	NBNS	92
117	15:00:43.198828	150.135.222.128	Intel_76:fc:06	1346	229.55.150.208	229.55.150.208	1345	UDP	219
118	15:00:43.424489	150.135.222.115	3com_c3:6e:09	netbios	150.135.223.255	150.135.223.255	netbios	NBNS	92
119	15:00:43.999997	150.135.220.242	00:19:d1:6e:27:48	1033	255.255.255.255	255.255.255.255	5200	UDP	63
120	15:00:44.175613	150.135.222.115	3com_c3:6e:09	netbios	150.135.223.255	150.135.223.255	netbios	NBNS	92
121	15:00:44.220171	3comEuro_98:03:4c	3comEuro_98:03:4c		Spanning-tree-(for	Spanning-tree-(for		STP	60
122	15:00:44.926653	150.135.222.115	3com_c3:6e:09	netbios	150.135.223.255	150.135.223.255	netbios	NBNS	92
123	15:00:45.159217	150.135.218.128	SunMicro_93:ca:02	33552	150.135.219.255	150.135.219.255	sunrpc	Portma	134
124	15:00:46.220191	3comEuro_98:03:4c	3comEuro_98:03:4c		Spanning-tree-(for	Spanning-tree-(for		STP	60
125	15:00:47.146181	150.135.220.205	Intel_79:89:a3	57642	150.135.223.255	150.135.223.255	netbios	NBNS	92
126	15:00:47.146249	10.20.30.205	Intel_79:89:a3	57642	10.20.30.255	10.20.30.255	netbios	NBNS	92
127	15:00:47.968348	150.135.222.10	Dell_e9:43:79	netbios	150.135.223.255	150.135.223.255	netbios	BROWSE	246
128	15:00:48.108852	150.135.218.31	WwPcbaTe_49:e0:0c	netbios	150.135.219.255	150.135.219.255	netbios	BROWSE	278
129	15:00:48.220224	3comEuro_98:03:4c	3comEuro_98:03:4c		Spanning-tree-(for	Spanning-tree-(for		STP	60
130	15:00:48.998952	150.135.220.242	00:19:d1:6e:27:48	1033	255.255.255.255	255.255.255.255	5200	UDP	63
131	15:00:50.221385	3comEuro_98:03:4c	3comEuro_98:03:4c		Spanning-tree-(for	Spanning-tree-(for		STP	60
132	15:00:50.273860	150.135.220.1	Netscree_42:54:09		224.0.0.13	224.0.0.13		PIMv2	60

```
⊞ Frame 73 (92 bytes on wire, 92 bytes captured)
⊞ Ethernet II, Src: Intel_76:fb:9d (00:0c:f1:76:fb:9d), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
⊞ Internet Protocol, Src: 150.135.216.236 (150.135.216.236), Dst: 150.135.219.255 (150.135.219.255)
⊞ User Datagram Protocol, Src Port: netbios-ns (137), Dst Port: netbios-ns (137)
⊞ NetBIOS Name Service
```

```
0000  ff ff ff ff ff ff 00 0c  f1 76 fb 9d 08 00 45 00   ........ .v....E.
0010  00 4e c8 00 00 00 80 11  90 a3 96 87 d8 ec 96 87   .N...... ........
0020  db ff 00 89 00 89 00 3a  42 09 83 f4 01 10 00 01   .......: B.......
0030  00 00 00 00 00 00 20 46  45 45 42 45 49 45 50 45   ...... F EEBEIEPE
0040  46 43 41 43 41 43 41 43  41 43 41 43 41 43 41 43   FCACACAC ACACACAC
0050  41 43 41 43 41 41 41 00  00 20 00 01               ACACAAA. . ..
```

Intel(R) PRO/100 VE Network Connection (Microsoft's Packet Scheduler) : <live capture in progress> File: C:... P: 135 D: 135 M: 0

Figure 12. Network traffic capture using Ethereal

Recall that the fundamental elements in network packet headers—an event time, a source IP address, a source MAC address, a source port number, a destination IP address, a destination MAC address, a destination port number, a protocol, and a packet length—are classified hierarchically based on SES methodology to evaluate network traffic activities. We set up Ethereal to capture only the packet elements listed above, and to capture network behaviors of the target network that make up the subnet of ACIMS lab. We notice that the number of events is very large; for example, the number of packet transmitted inside the subnet during one second is about fourteen hundred and twenty. Suppose that we need monthly information or even yearly information. Data captured during those time periods would include a tremendous number of events. It may take a very long time, or it may be impossible to evaluate that huge network traffic data because of memory overflows. But, fast and accurate network analysis is required for the network system manager to save budget and recover their systems from problems such as a hacker's intrusion, attacks, or a system down caused by viruses. That is the reason why we propose a new approach for analyzing network traffic behaviors quickly and efficiently based on the data engineering theorem. The data engineering methodology using the SES envisions efficient data management approaches for integrative system evaluations.

4.3 Instantiation (PES Generation)

This chapter illustrates the *Implement Level* in Figure 5. We use the SESBuilder for designing an SES (network traffic behaviors). The SESBuilder is an integrated tool to utilize the System Entity Structure (SES). The SESBuilder generates ontologies in XML schema format. The SESBuilder also creates XML instance files followed by SES pruning rules. The first step of performing the SESBuilder is writing a natural language formatted script that represents a behavior (ontology). Writing up ontologies by human-readable natural language is more convenient than writing up scripts by machine-readable languages. The SESBuilder has its own natural language interpreter, but we must follow its rules strictly. Figure 13 addresses the natural language which presents the SES shown in Figure 11.

A NetworkTrafficAnalysis can be HostBasedAnalysis or NetworkBasedAnalysis in analysis!

From the data perspective, a HostBasedAnalysis is made of PacketInfo, Time, Hosts, and Protocols!

The Time has a event_time!
The range of Time's event_time is string!

The Hosts has a num!
The range of Hosts's num is double with values 2!
From the mult perspective, a Hosts is made of more than one Host!
From the host perspective, a Host is made of a SrcHost, and DestHost!
From the compose perspective, A SrcHost is made of Addresses, and Ports!
From the compose perspective, A DestHost is made of Addresses, and Ports!

Addresses can be IPAddress, or MACAddress in mediaType!
The IPAddress has a ip_address!
The range of IPAddress's ip_address is string!
The MACAddress has a mac_address!
The range of MACAddress's mac_address is string!

From the mult perspective, Ports are made of more than one Port!
The Port has a port_number!
The range of Port's port_number is double!

Protocols can be TCP, UDP, HTTP, or FTP in protocol!
The Protocols has a protocol_type!
The range of Protocols's protocol_type is string!
The TCP has a protocol_name!
The range of TCP's protocol_name is string!
The UDP has a protocol_name!
The range of UDP's protocol_name is string!
The HTTP has a protocol_name!
The range of HTTP's protocol_name is string!
The ARP has a protocol_name!
The range of FTP's protocol_name is string!

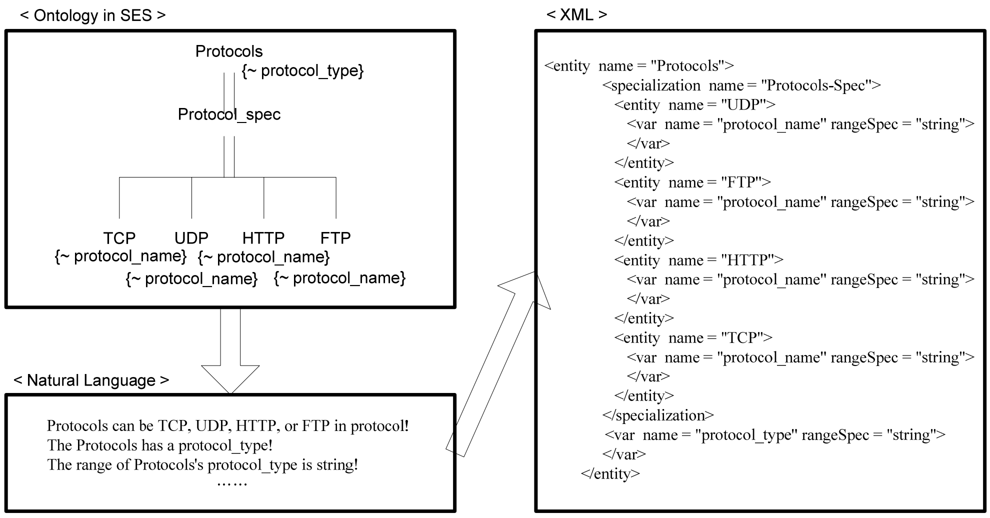
The PacketInfo has a packet_size!
The range of PacketInfo's packet_size is double!

Figure 13. Natural language representing the SES for network traffic behavior

The SESBuilder needs a natural language script as its initial input. The internal natural language interpreter verifies if an input natural language script follows its syntactic rules. Once a natural language script is verified, the SESBuilder generates an XML schema. The XML schema, including entities and attributes, represents the hierarchical SES tree structure. Processes of generating an XML schema from an SES and a corresponding natural language script in the SESBuilder are as given below:

1. Users write a natural language depicting a behavior
2. The SESBuilder interprets the natural language script
3. The SESBuilder generates an XML schema (SES) which represents the behavior

Figure 14 shows the relationships among the behavior in SES structure, the natural language, and the XML schema.



Protocols can be TCP, UDP, HTTP, or FTP in protocol!
The Protocols has a protocol_type!
The range of Protocols's protocol_type is string!
……

```
<entity  name = "Protocols">
        <specialization  name = "Protocols-Spec">
           <entity  name = "UDP">
              <var  name = "protocol_name" rangeSpec = "string">
              </var>
           </entity>
           <entity  name = "FTP">
              <var  name = "protocol_name" rangeSpec = "string">
              </var>
           </entity>
           <entity  name = "HTTP">
              <var  name = "protocol_name" rangeSpec = "string">
              </var>
           </entity>
           <entity  name = "TCP">
              <var  name = "protocol_name" rangeSpec = "string">
              </var>
           </entity>
        </specialization>
         <var  name = "protocol_type" rangeSpec = "string">
        </var>
     </entity>
```

Figure 14. Transformation from the SES to the XML code

An SES is a method to describe an ontology (behavior) conceptually. SESs (ontologies) represent real world states. A pruning operation instantiates SESs. The Pruned Entity Structure (PES) is a composition tree structure instantiated from SES, and it is used in simulation models and data engineering purposes. SESs are illustrated in XML DTDs or XML Schemas, but attributes' values of entities are not included in either XML DTSs or XML Schemas. The XML schema in Figure 14 shows entity names and attribute types. However, we need practical, usable objects that include real values. XML documents (PESs) include data values and information as well as entities. PESs (XML documents) are used in simulation models and data engineering purposes. Figure 15 illustrates how PESs are produced from an SES.

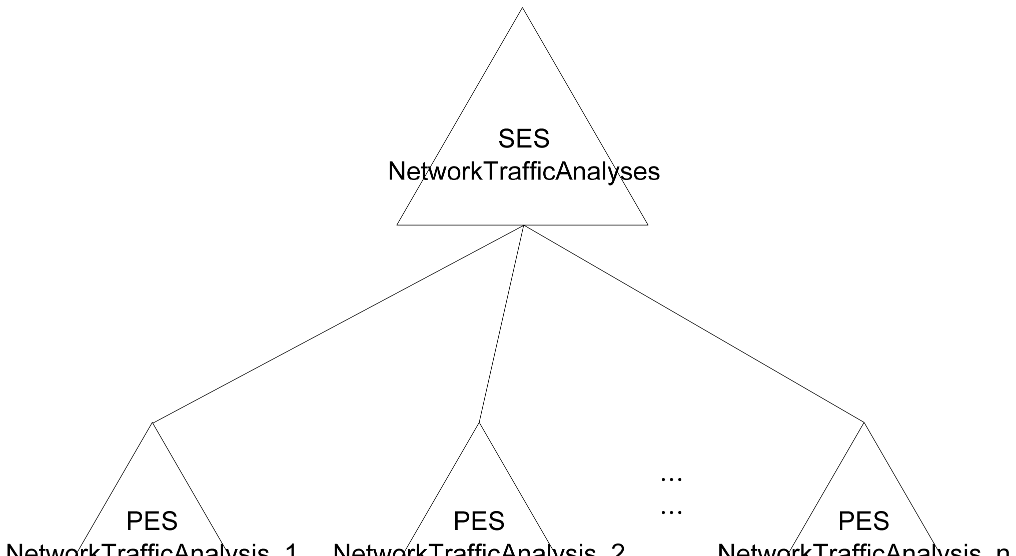


Figure 15. Multi-aspect for pruning of copies of its entity

Because networks are so active that the number of events, which is the number of generated packets, are enormous, they are not efficient to handle large amounts of data at one time. Therefore, we apply segmentation for pruning an SES. Every PES, from the *PES for NetworkTrafficAnalysis_1* to the *PES for NetworkTrafficAnalysis_n*, is segmented by time. For instance, if we monitor network behaviors during ten minutes and we prune by one minute segmentation, the resulting PES are ten XML instance files. Each of the XML instance files are treated in the modeling and simulation environment.

4.4 Automation

4.4.1 Generating new ontologies

Target network behavior analyses are defined by customers. We illustrate two cases: protocols analysis and network throughput measurement. The first analysis, evaluating the number of packets per protocols, requires two attributes of protocol names and identification numbers (ID). The second analysis, measuring network throughput, needs event times and packet sizes. This means that every analysis should have a different set of information. Keeping unnecessary information decreases computational power in both time (CPU) and size (memory). For speed and effectiveness, customers' requirements (pragmatic frames) need to create corresponding SESs which keep right entities and attributes. Consequently, users' target analyses must be modeled and simulated based on the new SES and their XML document instances (PESs). The unified processes, creating new SESs and setting up simulation environments dynamically according to users' requests, is our secondary objective in this study.

Once customers or users request a protocol usage analysis, a new SES is created automatically as given in Figure 16. The SES, *ProtocolAnalyses*, has multiple aspect of *ProtocolAnalysis*. The entity, *ProtocolAnalysis*, is composed of two entities, *ID* and *Protocol.* Users' requirements produce Java class files reflecting SESs. For preparation, we have a generic network behavior analysis class. New Java class files inherit from the generic network behavior analysis class. Objects of Java class files are assigned into a memory space automatically as soon as Java class files are produced. One instance of a class object file depicts one tuple of the SES for *ProtocolAnalyses*, a set of required entities. Multiple instances (tuples) of a class file compose a PES xml instance file for the practical purpose of modeling and simulation.

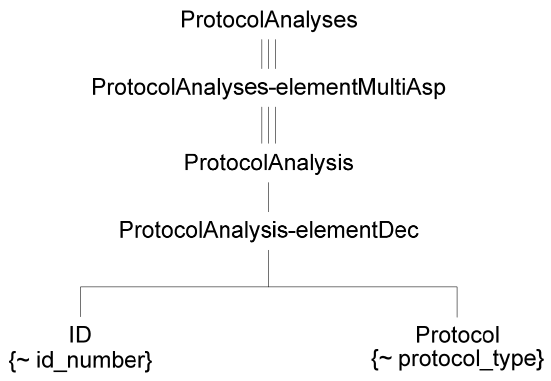


Figure 16. Ontology for protocol analyses in SES

Figure 17 shows an SES for the network throughput evaluation. The SES name is *ThroughputAnalyses*. *ThroughputAnalyses* has multi-aspect of *ThroughputAnalysis*.

ThroughputAnalysis is decomposed by *EventTime* and *PacketSize*. *EventTime* has an attribute, *event_time*, and *PacketSize* has an attribute, *packet_size*.

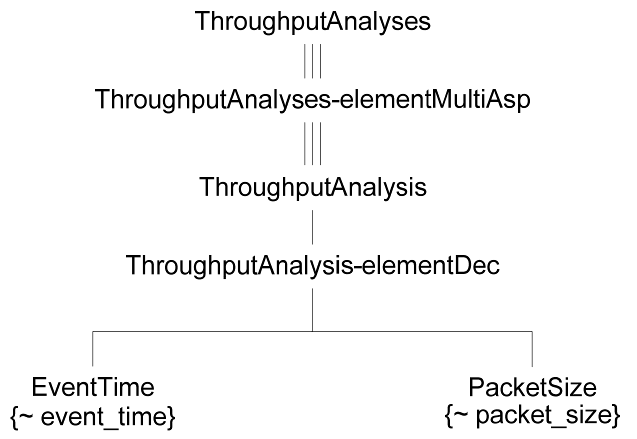


Figure 17. Ontology for throughput analyses in SES

4.4.2 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. First, we need to extract correct right data values from large PESs 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 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.

We design the SES, *NetworkTrafficAnalysis*, for generic purposes of network traffic behavior analyses as described in Figure 11. New SESs are generated to correspond to customers' requirements. The next step is producing new PESs based on a customized SES. First, we need to extract accurate data values from large PES instances (XML documents) of the SES, *NetworkTrafficAnalysis.* Consequently, new customized PESs are generated with the extracted attribute values. However, the problem is the case in which structures of two SESs, a source SES and a target SES, are different. The two SESs, the *NetworkTrafficAnalysis* and either the *ProtocolAnalyses* or the *ThroughputAnalysis*, have different structures. So, it is not possible to generate new PESs by transforming directly from source PESs. Alternatively, we must apply an approach, mapping operation. Mapping enables the retrieval of required data values from the *NetworkTrafficAnalysis* and the assigning of the data values to the other SES. Figure 18 illustrates the mapping process.

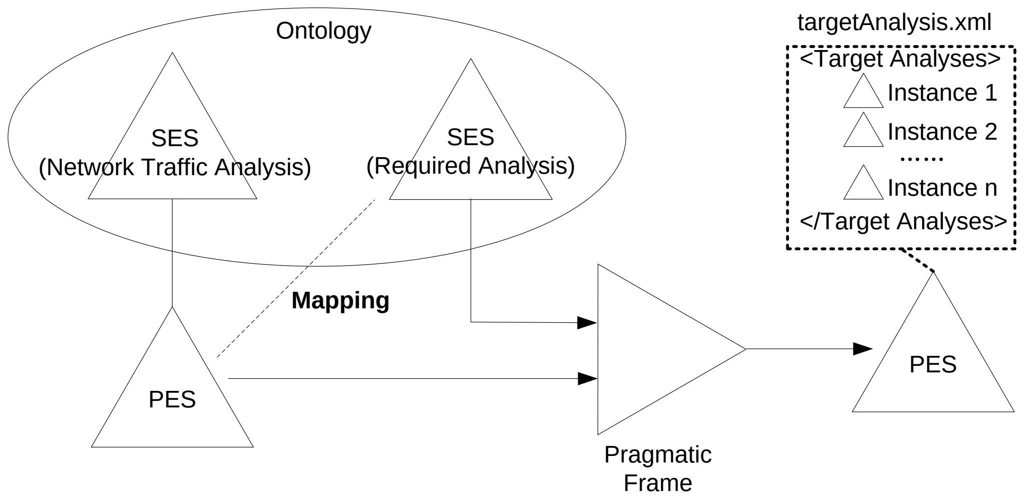
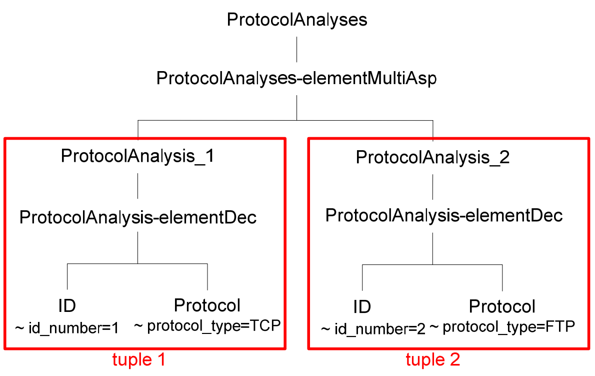
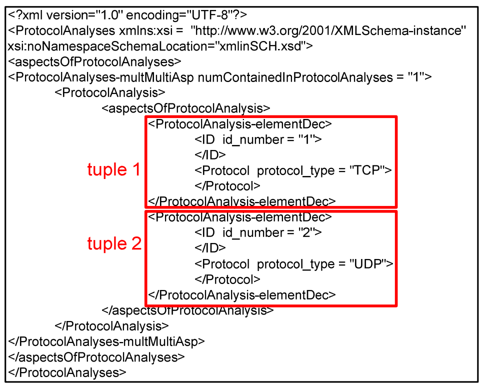


Figure 18. Mapping process

Outputs of mapping operations are PES xml instance files. Resulting PESs are composed of multiple instances of customers' required SES. One tuple includes attribute data of one packet transmission event in networks. For instance, there are two packet transmissions during monitoring time and a customer wants a protocol evaluation. Then, a resulting PES looks like the tree structure in Figure 19. Also the PES could be expressed by a XML instance file as shown in the right of Figure 19. Tuples have two attributes and their values. The XML instance file is used as a role of input source data for modeling and simulation purposes. Simulation models extract one tuple at a time and evaluate all the retrieved tuples at last for statistical or dynamic results. Processing one mass of a tuple instance at one time has advantages of speed and effectiveness against processing every small piece of the attributes.




tuple 1
tuple 2

```
<?xml version="1.0" encoding="UTF-8"?>
<ProtocolAnalyses xmlns:xsi =  "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="xmlinSCH.xsd">
<aspectsOfProtocolAnalyses>
<ProtocolAnalyses-multMultiAsp numContainedInProtocolAnalyses = "1">
        <ProtocolAnalysis>
                <aspectsOfProtocolAnalysis>
                    <ProtocolAnalysis-elementDec>
                            <ID  id_number = "1">
                            </ID>
                            <Protocol  protocol_type = "TCP">
                            </Protocol>
                    </ProtocolAnalysis-elementDec>
                    <ProtocolAnalysis-elementDec>
                            <ID  id_number = "2">
                            </ID>
                            <Protocol  protocol_type = "UDP">
                            </Protocol>
                    </ProtocolAnalysis-elementDec>
                </aspectsOfProtocolAnalysis>
        </ProtocolAnalysis>
</ProtocolAnalyses-multMultiAsp>
</aspectsOfProtocolAnalyses>
</ProtocolAnalyses>
```

Figure 19. PES for protocol analysis with two events

4.5 Automated Modeling and Simulation

Two processes, which are creating new ontologies and generating PESs through mapping operations, facilitate automated modeling and simulation. Figure 20 shows an overall architecture of these automated modeling and simulation processes. We could separate this overall process into the following four steps:

1. Capturing network behaviors, and generating PES instance files
2. Creating new ontologies according to users' requests
3. Mapping from the PESs of captured data to newly generated SES, and resulting PESs out
4. Modeling and simulation using PESs generated in the third step

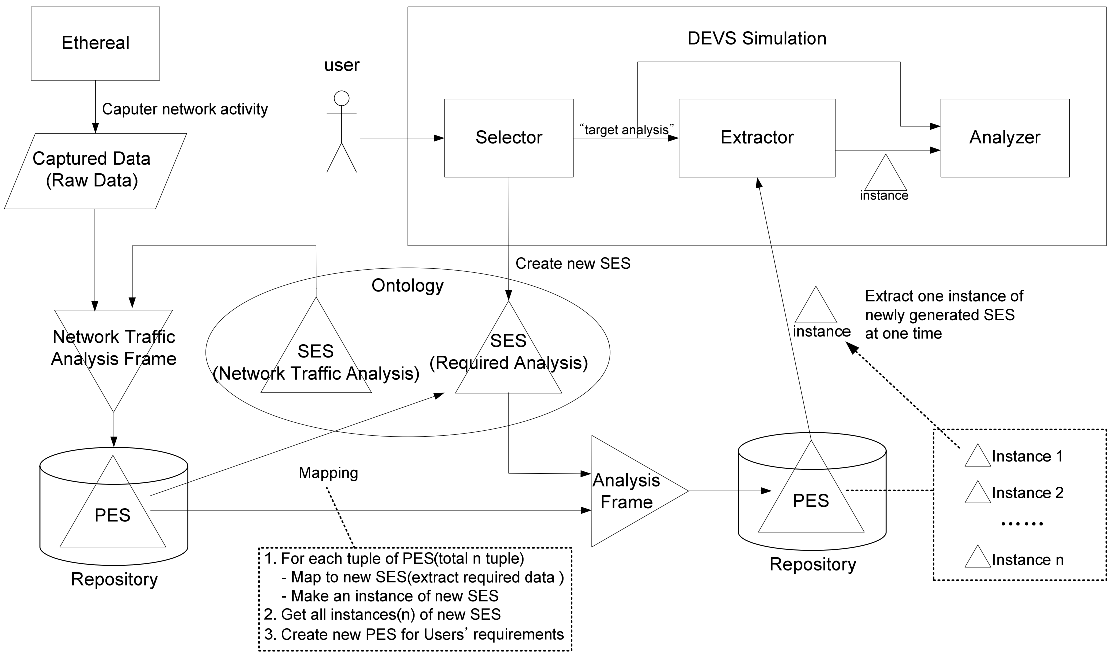

Ethereal
Caputer network activity
Captured Data (Raw Data)
user
DEVS Simulation
Selector
"target analysis"
Extractor
Analyzer
instance
Create new SES
Ontology
Network Traffic Analysis Frame
SES (Network Traffic Analysis)
SES (Required Analysis)
Extract one instance of newly generated SES at one time
instance
PES
Repository
Mapping
Analysis Frame
PES
Repository
Instance 1
Instance 2
……
Instance n
1. For each tuple of PES(total n tuple)
- Map to new SES(extract required data)
- Make an instance of new SES
2. Get all instances(n) of new SES
3. Create new PES for Users' requirements

Figure 20. Automated modeling and simulation

We need modeling and simulation to evaluate users' requested network behaviors. In this study, we build models and run simulation under the DEVSJAVA environment. DEVS atomic models and a DEVS coupled model are introduced in the next chapter.

CHAPTER 5. INTRUSION DETECTION SYSTEM

In this chapter, we discuss an advanced concept, 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 [45, 46].

Intrusion Detection Systems are systems designed to detect computer attacks. They monitor activities of computers and networks for attacks that are inevitable, despite security precautions. If attacks are discovered, intrusion detection systems can alert administrators, defend against the attacks, or provide information that may help prevent future attacks. Intrusion detection systems are not all equal in capabilities 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 intrusion detection system evaluations allow users to make informed decisions on what system to use and are extremely important for guiding research. Intrusion detection systems 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's article [47]. 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 repair damage, understand the attack mechanism, and reduce the possibility of future attacks of the same type. More advanced intrusion detection systems detect never-before-seen 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:

1. Problem 1. Limitation of data storage
    a. There are multitudes of events in networks and hosts
    b. Each event includes many attributes of packet information
2. Problem 2. Lack of analysis methods
    a. Difficulty of generating attacks
    b. Difficulty of implementing complete intrusion detection systems
3. Problem 3. Excessive resource consumption

a. Existing systems require huge computational resources in time (CPU) and space (memory).

A data engineering based modeling and simulation framework is intended to support testing and evaluation of network IDS. 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 KDD'99 dataset [42] was generated by MIT's Lincoln Lab for the purpose of testing network intrusion detection systems. The dataset includes various attack 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 data engineering based simulation framework for IDS. Our goal is to support a simulation framework for testing and evaluating network intrusion detection systems (IDS). Ontology/Data engineering methodology empowers our design to be efficient for managing and using large size data.

In this study, we build two IDS agent models, the LAND attack agent and the POD attack agent, and evaluate the two models. One advantage of SES/NZER is that it provides a simulation framework for testing IDSs. SES/NZER is available for IDS researchers to test their algorithms. IDS researchers build only their models corresponding to their IDS algorithms, and they request necessary attributes to evaluate their models. Other required models for simulations are provided. In addition to this scalability, SES/NZER should include more pre-defined models which are agents to detect various intrusions. Intrusions are classified into five kinds: Denial of Service attacks, User to Root attacks, Remote to Local attacks, Probes attacks, and Data attacks. If SES/NZER were capable of more functions, SES/NZER could give more convenience to users as a concrete tool. Intrusions are classified and described in Table 3. 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 if their IDS algorithms work well, necessary attribute values in network packet headers must be provided. According to researchers' target IDS algorithms, new SESs, which represent 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 11. Table 3 lists many attacks, and every attack in this table has different attack signatures. For example, detecting Apache2 attack needs to scrutinize in packet headers if http *GET* requests with the header "User-Agent: sioux\r\n" are over a certain number. A typical http request contains twenty or fewer headers in most systems. Therefore, a corresponding SES must hold three entities: protocol type, source IP address, and 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 expand the generic network behavior SES, which is shown in Figure 11, by including more entities such as Internet Header Length (IHL), Type of Service (TOS), Time to Live (TTL), header checksum, and other obtainable attributes from packet headers, into the SES. As a result, IDS developers may have better opportunities to evaluate precisely their algorithms.

Type	Name	Description
Denial of Service (DoS)	Apache2	Apache2 is an attack against an apache web server where a client sends a request with many http headers.
	ARPPoison	The goal of ARP Poison attack is to trick hosts on the same Ethernet by giving wrong MAC address for known IP address.
	Back	Back is an attack against an apache web server in which attackers submit requests with URLs containing many frontslashes(/).
	CrashIIS	Crash is an attack again an NT IIS web server. Attackers send malformed GET requests which crash the web server because GETs are part of IIS.
	DoSNuke	DoSNuke sends out of band data to port 139(NetBIOS), and crashs NT vitims.
	Land	Land is effective against older TCP/IP implementation. It sends spoofed SYN packet with the same source and destination IP address.
	Mailbomb	Mailbomb sends many messages to a server and overflows the server's mail queue.
	SYN Flood (Neptune)	TCP/IP implementation has a data structure to store pending connection, and the data structure is of finite size. SYN Flood fills the servers and the victims are not able to new coming connections.
	Ping of Death (POD)	Ping of Death attack affects older operating systems by sending oversized IP packets (Ping messages).
	Process Table	Process Table attack is against network services which fork or allocate new processes for incoming TCP/IP connections. So, it is possible to completely fill a target machine's process table and makes the machine be crashed.

	Selfping	Selfping is an attack in which normal users can remotely reboot systems with a single Ping command.
	Smurf	Smurf uses ICMP echo request packets to broadcast addresses from remote location. Machines that hear them respond packets. Finally, a victim is crashed.
	SSHProcesstable	SSHProcesstable lets victims fork so many children that the victim cannot spawn more processes.
	Syslogd	Syslogd attacks to remotely kill a syslogd services on Solaris servers.
	TCPreset	TCPreset listens for TCP connections to a victim and sends a spoofed TCP RESET packet to the victim. The victim terminates the TCP connection.
	Teardrop	Teardrop exploits a flow of old TCP/IP implementations, which do not properly handle overlapping IP fragments.
	Udpstorm	Udpstorm causes network congestions and slow down packet transmission time.
User to Root (U2R)	Anypw	Anypw attack allows an attacker to log in to a system without a password.
	CaseSen	CaseSen exploits the case sensitivity of the NT object directory. The attacker ftps three attack files to the victim: soundedt.exe, editwavs.exe, psxss.exe, and activate Trojan attack file.
	Eject	Eject exploits a buffer overflow of the 'eject' binary distributed with Solaris 2.5. It overwrites internal stack space of an eject program.
	Ffbconfig	Ffbconfig configures the Creator Fast Frame Buffer (FFB) Graphics Accelerator. It is possible to overwrite the internal stack space of the ffbconfig attack
	Fdformat	Fdformat attack formats diskettes and PCMCIA memory cards.
	Loadmodule	Loadmodule attack is used in SunOS 4.1 system to load two dynamically loadable kernel drivers into the currently running system. So, unauthorized users can gain root access on the local machine.
	Ntfsdos	Ntfsdos attack reboots the system from a floppy disk containing NTFSDOS.EXE.
	Perl	Perl attack exploits a bug in some Perl implementations, so that anyone with access to an account on the system can gain root access.
	Ps	Ps allows an attacker to execute arbitrary code with root privilege. Any users logged in to the system can

		gain unauthorized root privileges.
	Sechole	Sechole attack uploads test.exe and testfile.dll. The attacker runs test.exe. Then, the attacker is added to the Administrators group.
	Xterm	Xterm attack allows an attacker to execute arbitrary instructions with root privilege in Redhat 5.0 systems.
	Yaga	Yaga attack adds the attacker to the Domain Admins group by hacking the registry.
Remote to Local (R2L)	Dictionary	Dictionary attack tries to gain access to some machine by making repeated guesses at possible usernames and passwords with many services; telnet, ftp, pop, rlogin, and imap.
	FrameSpoofer	FrameSpoofer tricks a victim to believe he is viewing a trusted web site, but in actuality the page's main body is spoofed with a frame created by the attacker.
	Ftp-write	Ftpwrite takes advantage of a common anonymous ftp misconfiguration. An attacker will be able to add files and gain local access to the system.
	Guest	Because guest accounts are often left with no password or with an easy to guess password in most systems, Guest attack is one of the first and simplest vulnerabilities an attacker will attempt to exploit.
	HttpTunnel	HttpTunnel sets up and configures an http client to periodically query a web server. An attacker is able to "tunnel".requests for information through the http protocol.
	Imap	Imap attack exploits a buffer overflow in the Imap server of Redhat Linux 4.2. It allows remote attackers to execute arbitrary instructions with root privileges.
	Named	Named attack crash named server by requesting improper and malicious query on a TCP stream.
	Ncftp	Ncftp is an ascii UI ftp program for linux. Ncftp exploits the ability to get subdirectories recursively and creates new directories using the system command.
	Netbus	Netbus attack installs a trojan program and runs the Netbus server. Once Netbus is running, it acts as a backdoor attack. The attacker can access the machine using the Netbus client.
	Netcat	Netcat attack installs a trojan program and runs the netcat program on a specific port (53). So, the attacker can access the machine through the netcat port without a username or password.

	Phf	Phf attack abuses an improperly using CGI script to execute commands with the privilege level of the http server.
	PPmacro	PPmacro attack uses a trojan PowerPoint macro to read secret files. It saves secret files as ppt files, and posts them on a web.
	Sendmail	Sendmail attack sends a carefully crafted email message to a system, and attackers force sendmail to execute arbitrary commands with root privilege.
	SSHtrojan	SSHtrojan attack tricks the system administrator into installing a trojan version of the SSH program. This program allows an attacker to log in via ssh, with the login "monkey" and no password.
	Xlock	Xlock attack gains local access by fooling a legitimate user who has left their X console unprotected and obtains their password.
	Xsnoop	Xsnoop monitors keystrokes processed by an unprotected X server to gain information that can be used for local access.
Probes	Insidesniffer	Insidesniffer attacks a new machine to an inside ethernet hub, configured with an ip, and begins sniffing traffic.
	Ipsweep	Ipsweep attack monitor activities to determine which hosts are listening on a network
	Is_domain	Is_domain attack uses the "nslookup" command in interactive mode to "list" all machines in a given DNS domain. Attackers learn what machines connect to the DNS domain
	Mscan	Mscan is a probing tool that uses DNS zone transfers and scans IP addresses to locate machines.
	NTinfoscan	NTInfoScan is a NetBIOS based scanner. It scans and obtains share information: users, services running, and other information.
	Nmap	Nmap is a general-purpose tool for performing network scans.
	QueSO	QueSO is a utility used to determine what kind of machine and operating system exists at a certain IP adress.
	ResetScan	ResetScan sends reset packets to a list of IP addresses in a subnet to determine which machines are active
	SAINT	Security Administrator's Integrated Network Tool(SAINT) gathers information about remote hosts and networks by examining such network services as

		NFS, NIS, ftp, and other services.
	SATAN	SATAN is a previous version of the SAINT scanning program
Data	Secret	Secret attack maliciously transfers data which they have access to a place where it doesn't belong to

Table 3. Intrusion classification [48]

CHAPTER 6. Modeling and Simulation for Network Traffic Analysis

In this chapter, we illustrate how we build models and run simulations to evaluate customers' required network activity analyses. There are three major models: *Selector*, *Extractor* and *Analyzer*. The *Selector* model contains three functionalities: obtaining users' requests of target analysis, creating the new SES in a Java class format, and generating a new PES in XML format through mapping operations. The *Extractor* model reads events (packet transmissions in networks) from XML instance files (PESs) which are generated from the *Selector* model's process. The *Extractor* model obtains one packet event's information at one time. Packet event information includes attributes which are chosen by users through the *Selector* model. The *Extractor* model sends out messages which are SES tuple instances. The other model, the *Analyzer*, receives messages from the *Extractor* model and processes the messages. Once the *Analyzer* model receives all the messages from the *Extractor* model, the *Analyzer* model shows statistic results.

6.1 Selector Model

The *Selector* model is fundamental for automated context awareness. The functions of the *Selector* model are obtaining users' input requests, creating new SES tuple class object files, generating PES XML instance files, and notifying the users' target analyses to the other models that are connected with this *Selector* model. There are four states: *passive*, *Get_Req*, *Genr_PES*, and *out* states. In *passive* state, the *Selector* model waits for customers to activate it. Once the *Selector* model is activated, it becomes *Get_Req* state. During *Get_Req* state, several internal processes are performed. The *Selector* model invokes a GUI user input system. Customers could choose target analyses or select individual attributes for specific analyses. According to customers' requests, new SES tuple Java class files are created, and the class objects of the new SES tuples are consequently generated through the compiling process. After these processes are completed, the *Selector* model comes to the *Genr_PES* state, and new PES instance files are produced through mapping operations which are illustrated in chapter 4.4.2. At *out* state, the *Selector* model notifies users' selections to the other linking models to be ready. Figure 21 shows the state diagram of the *Selector* model.

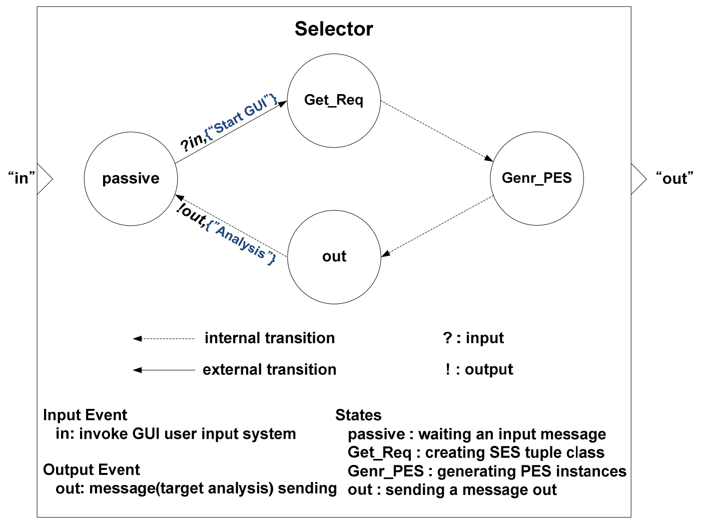

Selector
Get_Req
?in,{"Start GUI"}
passive
Genr_PES
"in"
"out"
!out,{"Analysis"}
out
internal transition
external transition
? : input
! : output
Input Event
in: invoke GUI user input system
Output Event
out: message(target analysis) sending
States
passive : waiting an input message
Get_Req : creating SES tuple class
Genr_PES : generating PES instances
out : sending a message out

Figure 21. State diagram for Selector model

We address several internal processes that should be performed in the *Get_Req* state. The Selector model never advances to the *Genr_PES* state from the *Get_Req* state until these processes are completed. Figure 21 illustrates the internal processes at *Get_Req* state. We could explain these internal processes just like the Finite Deterministic DEVS (FD-DEVS) [49, 50] performs. FD-DEVS is aimed towards development of DEVS models using a template-based design. FD-DEVS is a very convenient tool for building models in both an XML format and DEVSJAVA format. FD-DEVS generates models automatically according to users' requests.

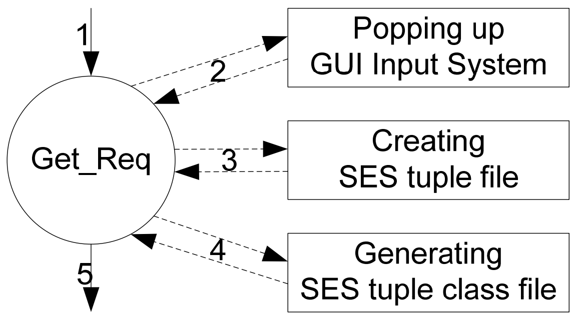


Figure 22. Internal processes at a state

6.2 Extractor Model

The *Extractor* model reads XML instance files which represent the PESs of ontologies for a customers' requested target network behavior analysis. Figure 23 shows the state diagram of the *Extractor* model. There are five states: *passive, ready, extract, generate,* and *end*. There exist two input ports ("*inAnal"* and "*in"*) and one output port ("*out"*) which have the purpose of sending/receiving messages. The *Extractor* model's initial state is *passive* and it comes to the *ready* state when it receives messages at the input port *inAnal*. The *Extractor* model is now ready to evaluate the analyses which are illustrated in the input messages. When an input, start extracting data, arrives at the input port *in*, the *Extractor* model comes to the *extract* state. During the *extract* state, the *Extractor* model retrieves information of one tuple in PES XML instance files at one time using the Document Object Model (DOM) library [51]. The model assigns a set of the attributes which are extracted from the PES XML files to an SES tuple class object. Next, the state changes to *generate* state. The *Extractor* model sends out messages which are the SES tuple class instances produced at the *extract* state. If there exist no more data to read in PES XML files at *extract* state, the model's state changes to *end* state, and the model transmits an "end" message. Consequently, the *Extractor* model finally turns back to the *passive* state.

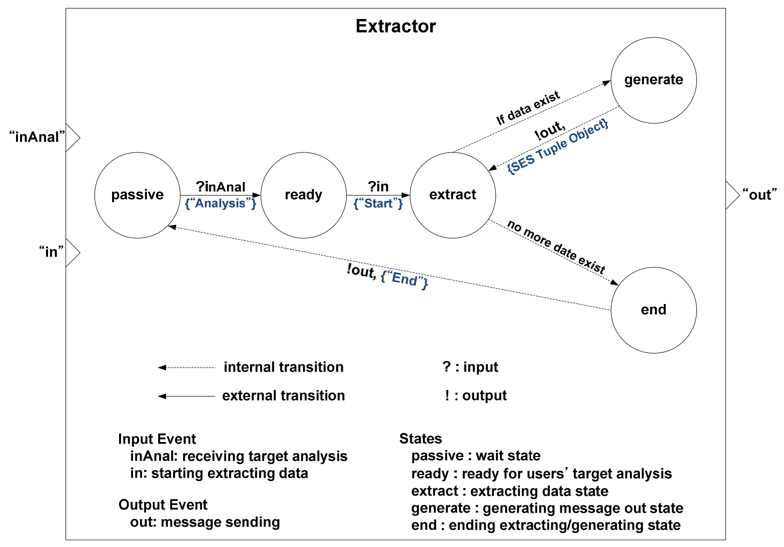


Figure 23. State diagram for Extractor model

6.3 Analyzer Model

The *Analyzer* model evaluates messages received from an input port and shows the results statistically. Figure 24 shows the state diagram of the *Analyzer* model. The initial state is the *passive* state, and the time advance is set to infinity in this state. If the *Analyzer* model receives a message which addresses a target analysis, at the input port *inAnal*, the state changes to *ready* state. When a message arrives at the input port *in*, the model comes to the *busy* state. During the *busy* state, the *Analyzer* model receives messages which describe SES tuple instances. Consequently, the target evaluation is being processed, and the *Analyzer* model waits for input messages at the *busy* state. Once the *Analyzer* model receives an "*end*" message instead of SES tuple instance messages, the *Analyzer* model comes to the *end* state and sends the results in statistical outputs. The outputs are shown in graphical user interface chart diagrams. For the GUI chart diagrams, we use the JFreeChart library [52].

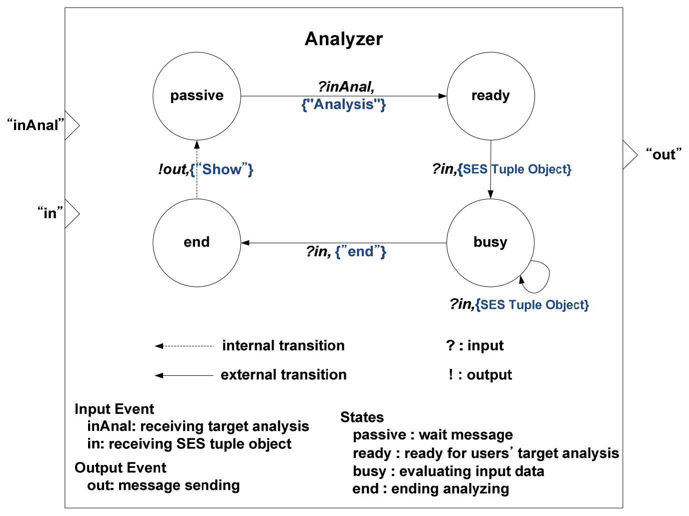


Figure 24. State diagram for Analyzer model

6.4 Coupled Model for Network Traffic Analysis

Recall that basic models may be coupled in the DEVS formalism to form a coupled model. A coupled model is the major class which embodies the hierarchical model composition constructs of the DEVS formalism. A coupled model is defined by specifying its component models, called its components, and the coupling relations which establish the desired communication links.

In this study, the basic models are the *Selector* model, the *Extractor* model, and the *Analyzer* model. We make a coupled model, *NetworkTrafficAnalysis*, by linking the Extractor model and the Analyzer model. The output port of the *Extractor* model and the input port of *Analyzer* model are connected to each other for sending/receiving messages. The purpose of this coupled model, *NetworkTrafficAnalysis*, is evaluation. Then, we make a coupling between the *Selector* model and the coupled model, *NetworkTrafficAnalysis*. The output port ("*out*") of the *Selector* model links to an input port of the *NetworkTrafficAnalysis* model. As soon as the *NetworkTrafficAnalysis* receives input messages, the *NetworkTrafficAnalysis* model delivers the input messages to its component models, the *Extractor* model and *Analyzer* model. Messages from the *Selector* model enable the *Extractor* and the *Analyzer* to be ready to evaluate network traffic behaviors. For instance, if an output message of the *Selector* model represents protocol analysis, the *Extractor* model and the *Analyzer* model are set up for analyzing protocols. Otherwise, if a user grants throughput analysis input to the *Selector* model, then the simulation environment is set up for network throughput analysis. The automated simulation environment is set up by the following steps:

1. The *Selector* model acquires users' requests.
2. Internal processes of the *Selector* model produces PES XML instance files according to users' requests.
3. The *Selector* model sends a message (target analysis) out to the *Extractor* model and the *Analyzer* model.
4. The *Extractor* model and the *Analyzer* model are ready for the target analysis.

The *Extractor* model reads the PES XML instance files including network behavior information such as event times and protocols, and the *Extractor* model sends a message out to the *Analyzer* model. The *Analyzer* model receives messages from the *Extractor* model, and it evaluates the messages. Finally, the *Analyzer* model concludes statistical simulation results to easy-to-read graphical user interfaced charts as soon as all the data is analyzed. Figure 25 shows the DEVS coupled model and its components.

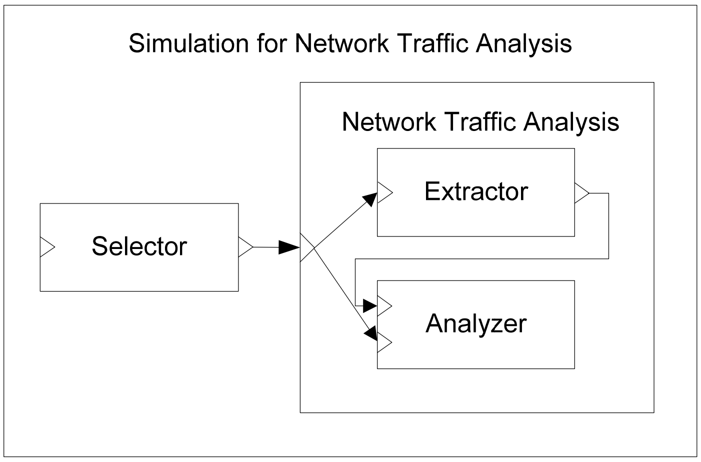


Figure 25. DEVS coupled model and its components

6.5 Experimental Results for Network Traffic Analysis

Recall that we monitor network behaviors in a subnet of the Arizona Center for Integrative Modeling and Simulation (ACIMS) lab in the department of electrical and computer engineering, the University of Arizona for this study. We monitor one-day network behavior from Jan 16 9:00 AM to Jan 17 9:00 AM. The total number of events is 2,045,699. In other words, there is an average of 1420 packet transmissions per second. Network administrators of large organizations such as companies, governments, and universities may have large amounts of data than the amounts of data that could be captured in the ACIMS lab. This enormous data size leads to memory overflows and degradation of computational powers. Therefore, automated, efficient, and fast ways of analyzing network behaviors and detecting system problems become more important as network activity grows.

The original captured data size in text file format is 288MB. The size of XML files which is transformed from the captured data by Ethereal is 5.43GB because of additional tags that represent the SES (behavior) as a tree structure. XML format is easier to read and understand than text format, but 5.43GB of XML files is about twenty times over the size of the 288MB text files. However, we prune the 5.43GB sized XML files according to the users' target analysis. In this study, we test two cases: a protocols usage analysis and a network throughput analysis. For the first case study of protocol analysis, we gain 323MB XML files which have the two attributes of a packet ID and a protocol name. For the other case study of throughput analysis, we obtain 385MB XML files having an event time and packet size. Either 323MB or 385MB is still larger than the original text file size of 288MB. However, the XML files keep only necessary data for the analyses. Also, the tags in the XML files enable easy and fast access to attribute values.

Figure 26 represents the DEVSJAVA simulation environment. We design three major atomic models: the *Selector*, the *Extractor*, and the *Analyzer*. The coupled model, *NetworkTrafficAnalysis*, includes the *Extractor* and the *Analyzer* models. The output port of the *Selector* model and the input port of the coupled model, *NetworkTrafficAnalysis*, are connected for message passing. The coupled model, *NetworkTrafficAnalysis*, distributes its input messages to its components, the *Extractor* and the *Analyzer*. Also, in the coupled model, the *Extractor* and the *Analyzer* models are connected each other so that the simulation runs extracting data from PES XML instance files and evaluating the data. For the protocols/services analysis purpose, the data elements extracted from the PES XML files are packet IDs and protocols. For the other purpose of network throughput analysis, event times and packet sizes are retrieved from the database XML files. The others elements in XML files, such as source IP address, source port number, destination IP address, and destination port number, are not considered to be extracted as a result of fast and effective data processing.

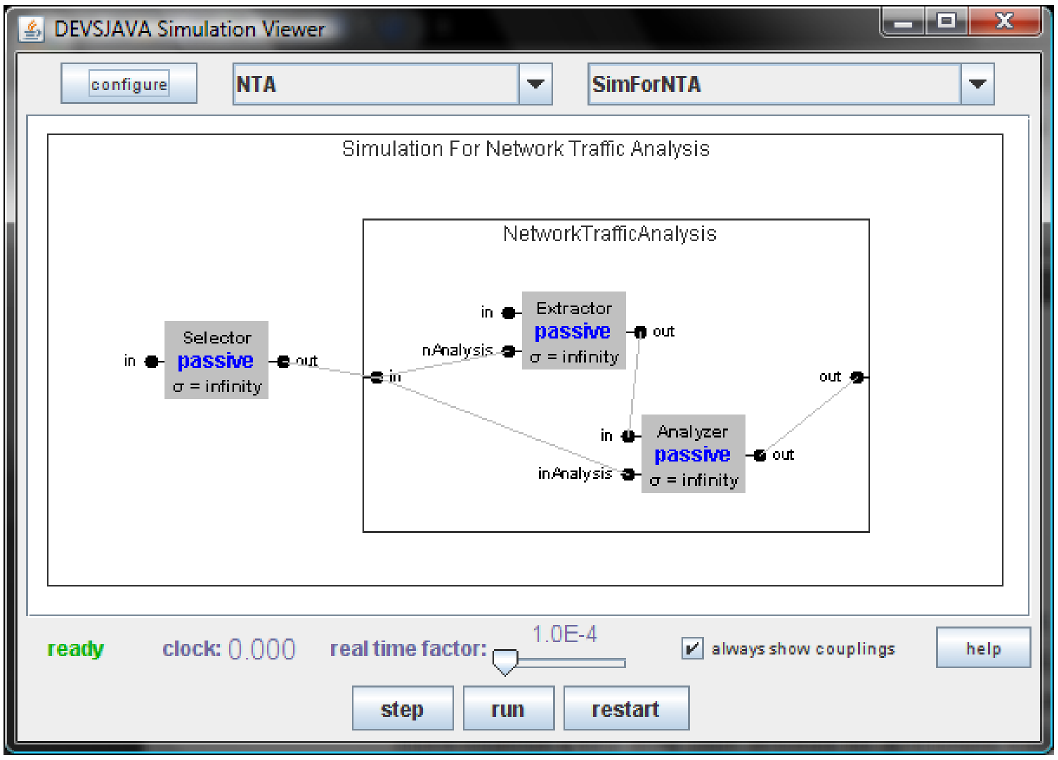


Figure 26. DEVSJAVA simulation

The statistical results are shown in GUI reporting windows in forms of both the charts analysis and the text analysis method. Figure 27 shows the results of the protocol analyses. When a simulation is done, a GUI window is invoked to represent statistical results in text. Text results illustrate protocols and the number of events per protocols. There are 71 kinds of protocols in monitored network activities. The most visible protocol is Address Resolution Protocol (ARP) with 1,742,107 events. This is about 85 percent of the total packet transmissions. ARP is a protocol for mapping an Internet Protocol address (IP address) to a physical machine address that is recognized in the local network. As well as text results, two kinds of charts are shown as given in Figure 27. Bar charts can easily illustrate the number of events per protocols. Pie charts show intuitive comparisons between protocols.

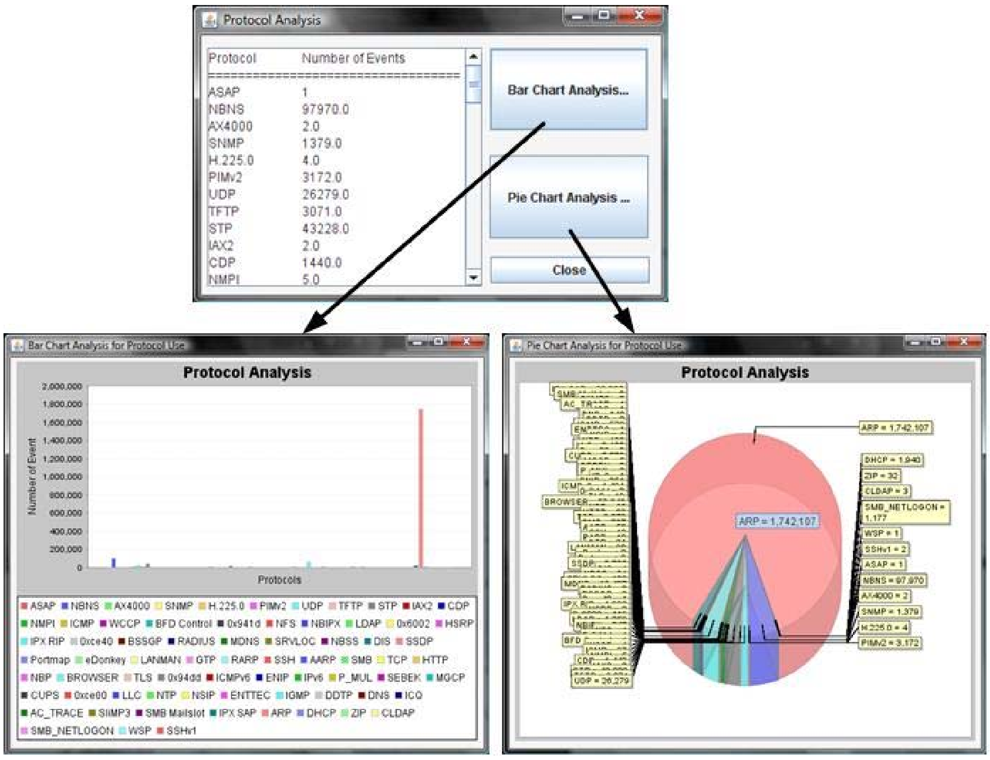

```
Protocol       Number of Events
===================================
ASAP           1
NBNS           97970.0
AX4000         2.0
SNMP           1379.0
H.225.0        4.0
PIMv2          3172.0
UDP            26279.0
TFTP           3071.0
STP            43228.0
IAX2           2.0
CDP            1440.0
NMPI           5.0
```
Protocol Analysis
Bar Chart Analysis...
Pie Chart Analysis ...
Close
Bar Chart Analysis for Protocol Use
Protocol Analysis
2,000,000
1,800,000
1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
Number of Event
Protocols
ASAP NBNS AX4000 SNMP H.225.0 PIMv2 UDP TFTP STP IAX2 CDP
NMPI ICMP WCCP BFD Control 0x941d NFS NBIPX LDAP 0x6002 HSRP
IPX RIP 0xce40 BSSGP RADIUS MDNS SRVLOC NBSS DIS SSDP
Portmap eDonkey LANMAN GTP RARP SSH AARP SMB TCP HTTP
NBP BROWSER TLS 0x94dd ICMPv6 ENIP IPv6 P_MUL SEBEK MGCP
CUPS 0xce00 LLC NTP NSIP ENTTEC IGMP DDTP DNS ICQ
AC_TRACE SIIMP3 SMB Mailslot IPX SAP ARP DHCP ZIP CLDAP
SMB_NETLOGON WSP SSHv1
Pie Chart Analysis for Protocol Use
Protocol Analysis
ARP = 1,742,107
DHCP = 1,940
ZIP = 32
CLDAP = 3
SMB_NETLOGON = 1,177
WSP = 1
SSHv1 = 2
ASAP = 1
NBNS = 97,970
AX4000 = 2
SNMP = 1,379
H.225.0 = 4
PIMv2 = 3,172
ARP = 1,742,107
BROWSER
UDP = 26,279

Figure 27. Simulation result for protocol analysis

We also evaluate network throughput. To analyze network throughput (bytes per minutes), we require two attributes: event time and packet size. Figure 28 helps to understand network throughput variations. The results show that there are no abrupt changes during one day. The network keeps between 80,000 bytes to 180,000 bytes per minute. So, a network manager may conclude that the maximum bandwidth of 200,000 bytes per minutes is big enough to maintain this network.

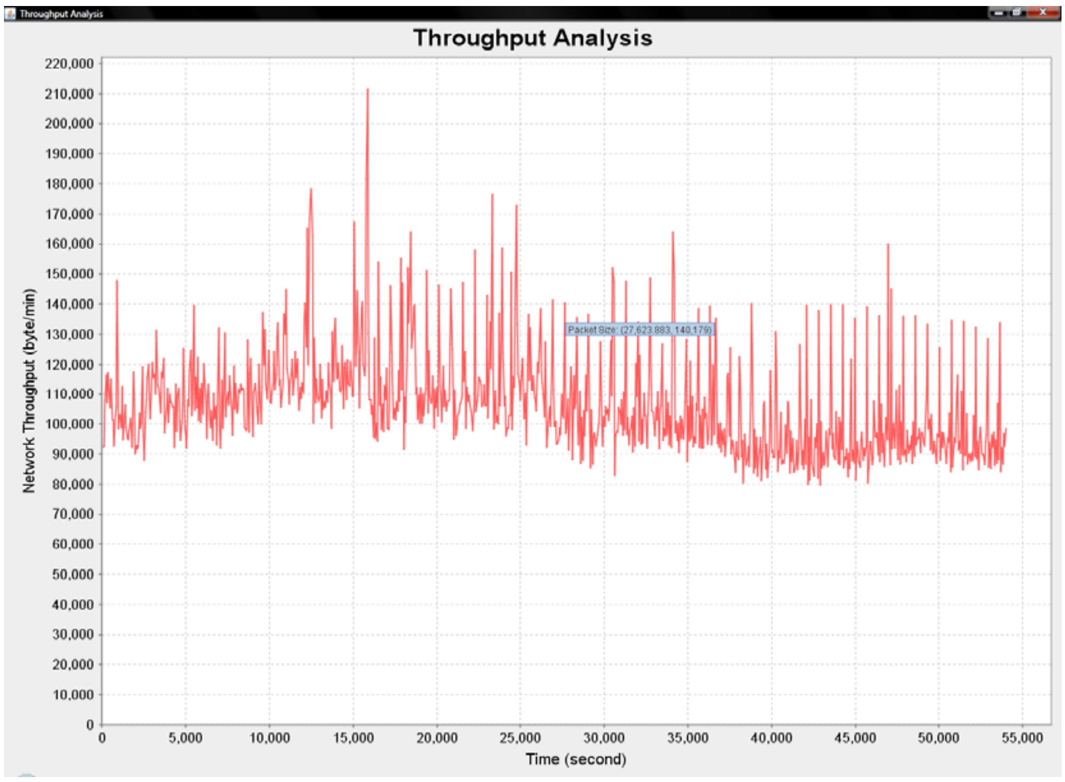


Figure 28. Simulation result for throughput analysis

The advantages of SES/NZER are easy and fast information capturing of large amounts of data, a fast response time, and a user centric schema. The fact that SES/NZER is developed based on ontology/data engineering methodology represented by SES theory facilitates users reading, understanding, and manipulating the network data easily. The user-friendly graphical chart gives customers, who are network administrators, general ideas with respect to their requests such as protocols analysis and network throughput evaluation. In this study, we have illustrated the strength of using the SES to represent a large data set consisting of elementary units through the concept of multiAspect. As we have shown, the SES concept of pruning employs aspects, multiAspects and specializations to allow very flexible specification of subsets of a given data set, and aggregation operations on them. Aggregation is a form of abstraction commonly employed in modeling in disciplines ranging from physics to economics [17]. The aggregation restructuring concepts give a better idea of how to select subsets of event data and how to aggregate the subsets together to integrate various ubiquitous systems in real-world applications.

CHAPTER 7. DISCUSSIONS

7.1 Comparisons of SES/NZER with Ethereal

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. We use Ethereal to capture network behaviors in this study. The captured data is evaluated by SES/NZER. Table 4 illustrates comparisons of analyzing manners between Ethereal and SES/NZER.

	Ethereal	**SES/NZER**
Functionality	Protocol Analysis Throughput Analysis Service Analysis And more	Protocol Analysis Throughput Analysis User-selected and combined Analysis
Analyzing Methods	Graphical charts Text Analyses	Graphical charts Text Analyses
Complexity	Complicated • Hard to learn • Hard to evaluate	Simple • Easy to learn • Easy to understand
Locality	Local machine only • Local machine : Monitor, capture, and analysis	Potentially distributed Environment • Local machine: monitor, capture • Remote machine: analysis

Data Size	Big complete data for every analysis	Specified small data for each analysis
Modularity	One process	Several individual processes
Scalability	Complete system Not good for flexibility	Possible to add new analyses Interoperability with XML-related systems

Table 4. Comparisons between Ethereal and SES/NZER

The first comparison is functionality. Ethereal has been supported by many network professionals, so it has many functions, such as protocol analysis, throughput analysis, and other statistical analyses. SES/NZER focuses on customers' specific requests. In this study, we study two cases: protocol analysis and throughput analysis. But, SES/NZER is widely open to be developed for further requests. The second aspect to comparatively analyze is the methods. Both Ethereal and SES/NZER have two kinds of resulting methods, graphical charts and textual analyses. Ethereal is like a two-sided coin. Ethereal is very powerful but also very complicated. Ethereal requires an initial learning curve. SES/NZER is simple and customized for target analyses, so it is easy to learn and understand. Ethereal is one complete tool, and it is limited to running on local machines. On the other hand, SES/NZER is a combination of individual processes such as monitoring and capturing processes and analyzing process. As a result, it has the potential to extend to a distributed environment. Monitoring and capturing network activities could be performed in local hosts, and analyzing network behaviors could be evaluated in remote hosts. SES/NZER is intentionally designed for distributed simulations. The DEVS modeling shown in Figure 25 has two components: the *Selector* model and the

NetworkTrafficAnalysis model. We could distribute the models into multi-servers. A distributed simulation could be performed by message-passing methods among servers. A web service middleware, DEVS Service Oriented Architecture (DEVS/SOA) [53, 54], facilitates in distributing workloads and scaling to handle multiple customers. The ways of accessing data are different. Ethereal uses complete data for every analysis. Accessing a big data set requires memory overhead and inefficient computational power. SES/NZER needs small sized data for each analysis. It takes time initially to generate user specific pruned data, but using compact data is a fast and effective approach. This is the most important advantage of a SES based system. Figure 29 shows these different data accessing methods.

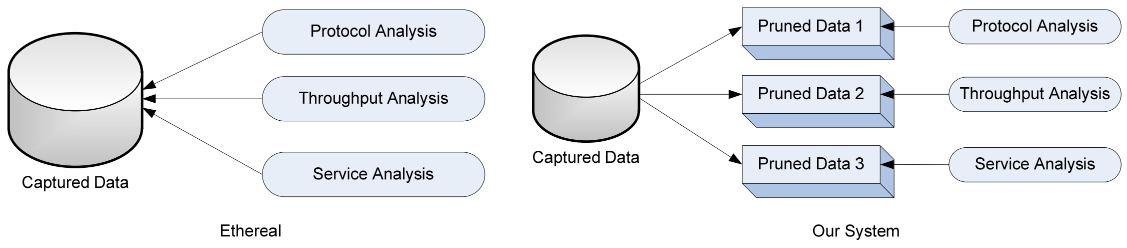


Figure 29. Data access methods

We measure system memory (RAM) usages and execution times of both Ethereal and SES/NZER. We use a half-day, one day, and two days of data to evaluate system performance variations. Table 5 shows measurements of memory uses and execution times for network protocol analyses. Table 6 illustrates experimental results for throughput evaluations.

	Ethereal			**SES/NZER**		
	Half day	One day	Two days	Half day	One day	Two days
Loading time	1 min 18 sec	2 min 28 sec	N/A	5 min 28 sec	10 min 44 sec	20min 59sec
Number 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 sec	50 sec	N/A	5 min 29 sec	10 min 58 sec	22min 59sec

Table 5. 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 sec	2 min 28 sec	N/A	5 min 32 sec	11 min 27 sec	22min 14min
Number 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 sec	55 sec	N/A	5 min 17 sec	9 min 56 sec	22min 13min

Table 6. Memory usages and execution times for throughput analysis

The loading time of Ethereal refers to the time of invoking the captured data file. The loading time of SES/NZER is a time of generating PES XML document files with regards to users' requests. SES/NZER takes a longer time for loading data to evaluate than Ethereal. Also, Ethereal is faster to analyze data than SES/NZER. We notice that both loading time and analyzing time increase linearly corresponding to total numbers of events during capturing period. Figure 30 illustrates comparisons between data flow of Ethereal and data flow of SES/NZER.

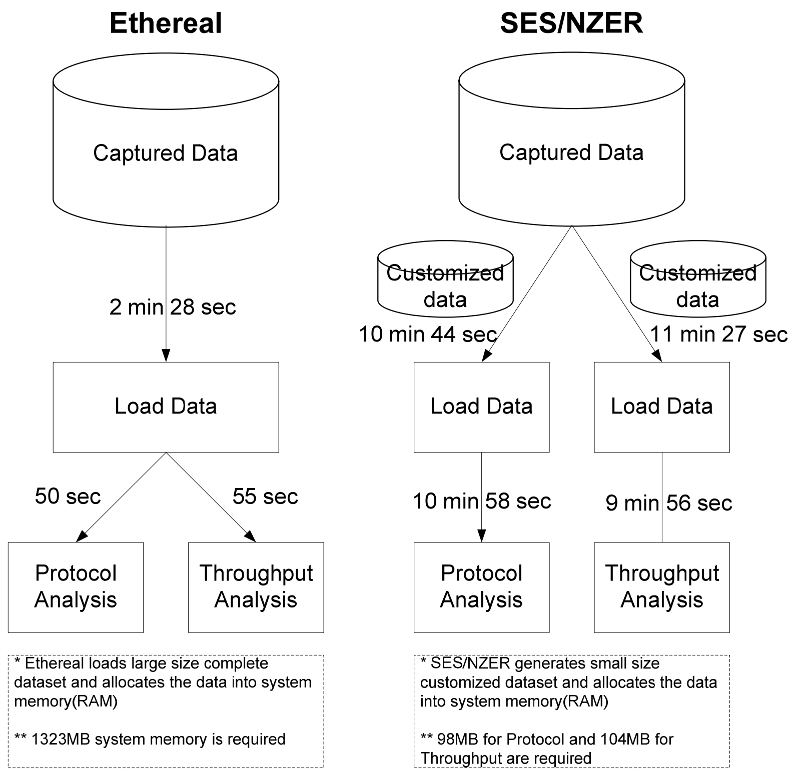

Ethereal
SES/NZER
Captured Data
Captured Data
~~Customized~~ data
~~Customized~~ data
2 min 28 sec
10 min 44 sec
11 min 27 sec
Load Data
Load Data
Load Data
50 sec
55 sec
10 min 58 sec
9 min 56 sec
Protocol Analysis
Throughput Analysis
Protocol Analysis
Throughput Analysis
* Ethereal loads large size complete dataset and allocates the data into system memory(RAM)
** 1323MB system memory is required
* SES/NZER generates small size customized dataset and allocates the data into system memory(RAM)
** 98MB for Protocol and 104MB for Throughput are required

Figure 30. Data flow comparisons

Table 5 and Table 6 indicate that Ethereal is faster than SES/NZER. However, Ethereal is a complete tool, so it should be run on a single machine only. On the other hand, SES/NZER is scalable to distributed environments. 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 number of servers. Ultimately, 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 a system memory (RAM). As data size increases, the memory requirement of Ethereal increases linearly. However, SES/NZER needs 98MB of a system memory for half-day data, and the memory requirement of SES/NZER never increases in correspondence to source data sizes. SES/NZER fragments source data into multiple numbers of small size datasets. SES/NZER allocates one segmented dataset to system memory. Once evaluating the dataset is completed, SES/NZER frees the dataset from system memory, and, consequently, it loads another dataset into system memory. Therefore, SES/NZER holds only one small-sized dataset during simulation time frame, and the memory requirement never increases. SES/NZER keeps the system stable. For two-days captured data, Ethereal cannot load data and consequently cannot analyze the network activities. Ethereal is shut down due to memory overflow problems. On the other hand, SES/NZER can evaluate network behaviors although it takes time.

7.2 Problem Statements

The fact that SES/NZER is more efficient in system memory requirement than Ethereal facilitates SES/NZER analyzing large amount of data. However, SES/NZER is weak in evaluation speed. One solution to achieve feasible speed-up and efficiency is parallel processing. This is the reason why developing a web-based distributed SES/NZER is a promising research area of network analysis fields. Parallel processing consists of dividing data into two or more smaller datasets, assigning datasets into multiple processors, and processing multiple datasets in multiple processors simultaneously. Divide and conquer (D&C) is an important algorithm design paradigm. Divide and conquer was first introduced by Karatsuba [55] as an algorithm for multiplying two n-digit numbers with an algorithmic complexity $O(n)$ on $n^{\log_2 3}$. But, the divide and conquer scheme is widely used in parallel processing designs for reducing complexity of processors. Divide and conquer 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 smaller problems are combined to produce a solution for the original problem. Figure 31 shows a divide and conquer scheme for SES/NZER.

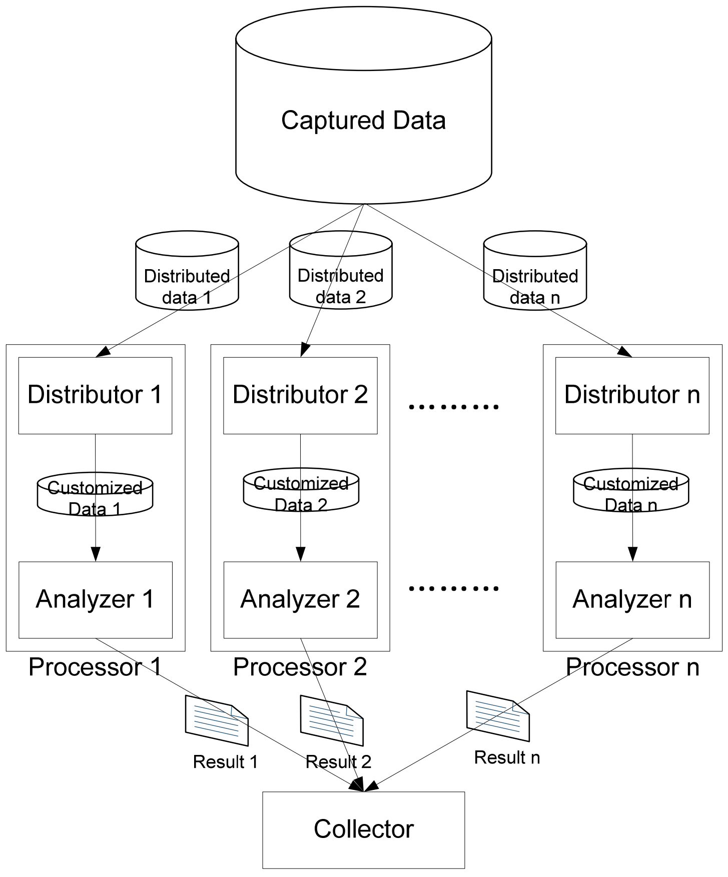


Figure 31. Divide and conquer SES/NZER

The first step is the dividing process. 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 the analyzed results of processors are integrated together at last. This integrating of all the results and concluding with a final output is the conquering process. This divide and conquer approach requires not only segmentation overheads for dividing data but also communication overheads for conquering all the results. Even though there are overhead disadvantages, this method includes two strengths which overcome the disadvantages. One advantage is that this approach enables applications, which need to process 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 divide and conquer approach may be applied to distributed environments. Processors are deployed into multiple machines which 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 each processor's activities never affect other processors' behaviors. In this study, we use web service schemes over Service Oriented

Architecture (SOA) to construct distrusted environments. This web-based distributed simulation increases independency and decreases complexity in each host. Table 7 illustrates comparisons between SES/NZER and a Web-based 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

Table 7. SES/NZER Vs. Distributed SES/NZER

CHAPTER 8. WEB-BASED DISTRIBUTED SES/NZER

8.1 Design Issues

In this study, 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. And, intrusion detection systems are evaluated for specialized cases. Figure 32 represents the hierarchical system structure.

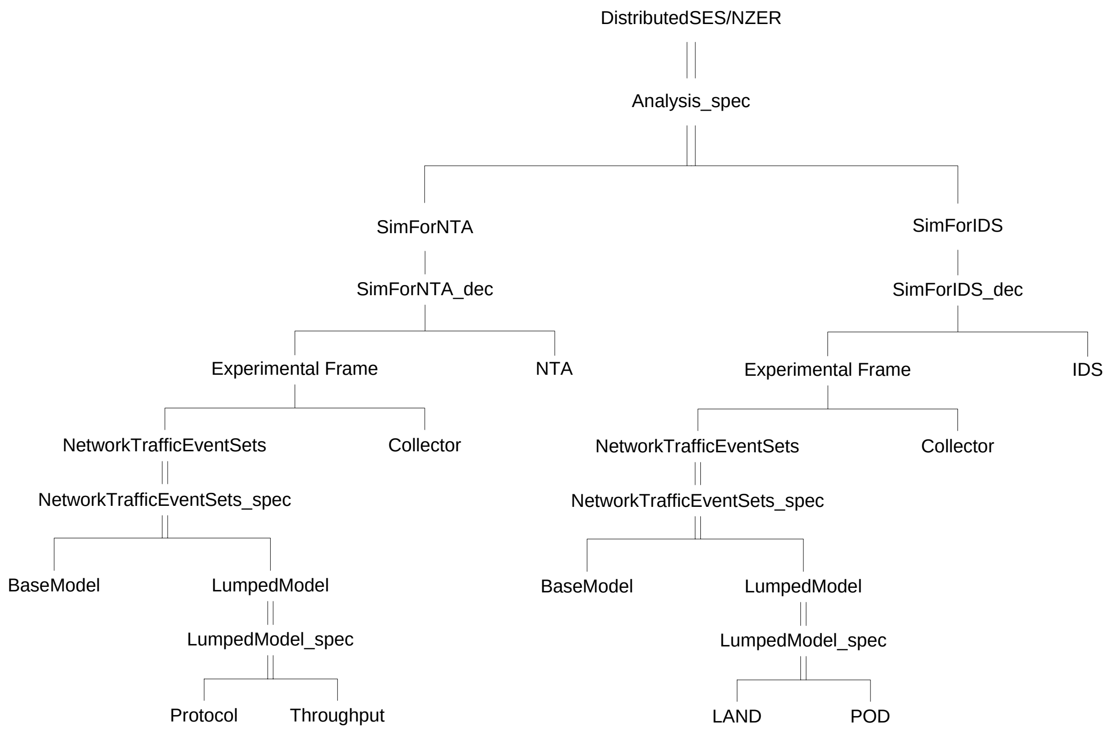


Figure 32. Distributed SES/NZER system hierarchy

A Web-based distributed SES/NZER fulfills either analyzing generic network traffic activities (protocol analysis or throughput analysis) or evaluating an intrusion detection system (LAND or POD). A complexity constraint is that modeling is severely limited [17]. 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. Computers continue to become faster and increase in memory, but 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 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 modeling and simulation project. Figure 33 shows a pair of models involved. They are base and lumped models in an experimental frame.

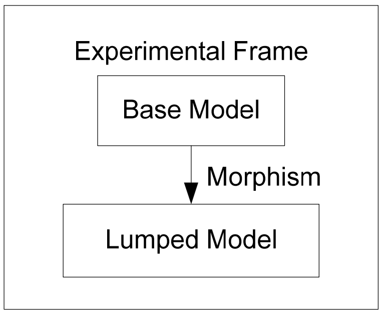
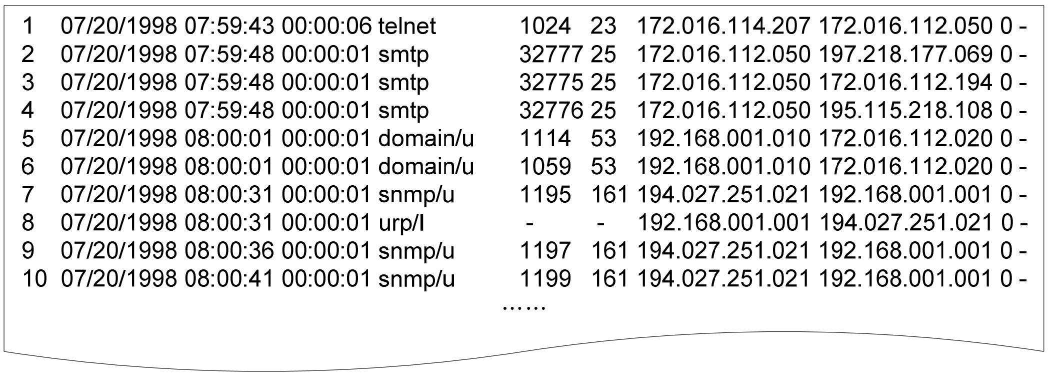


Figure 33. Base/lumped model equivalence in experimental frame

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 descriptions 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 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 (CPU) and size (memory).

8.1.1 Pragmatic Frames (Lumped Models)

We design the SES for illustrating the generic network behaviors in Figure 11. This SES represents based models of both simulation for network traffic analysis and simulation for intrusion detection systems (IDS). For the use of generic network traffic analysis simulation, we monitor network activities and capture the fundamental packet information in ACIMS lab using the Ethereal. Unlike generic network behavior analyses, source data for IDS simulation must include attack packet transmissions as well as normal packet transmissions. But, generating attack packets is strictly prohibited even if it is for academic research purposes. Therefore, for the purpose of intrusion detection system simulation, we use a KDD’99 dataset [42]. The MIT Lincoln lab supported by the DARPA project [56] simulated and generated a network traffic dataset, including attacks, in 1998. This dataset has been widely used in the area of computer network intrusion detection system research and is now regarded as the standard. Also, it is well-known by the name, KDD’99 dataset, because Knowledge Discovery and Data Mining [57] processed the network traffic data generated by MIT Lincoln lab and opened a contest. Many network researchers and artificial intelligent researchers use this dataset for their intrusion detection system. The dataset includes two weeks (five days/week) simulation data. Every day data set is huge, e.g., the first week’s Monday data has 60,000 events. According to the SES in Figure 11, the KDD’99 dataset is re-structured. Figure 34 represent KDD’99 dataset.



```
1   07/20/1998 07:59:43 00:00:06 telnet      1024  23  172.016.114.207 172.016.112.050 0 -
2   07/20/1998 07:59:48 00:00:01 smtp        32777 25  172.016.112.050 197.218.177.069 0 -
3   07/20/1998 07:59:48 00:00:01 smtp        32775 25  172.016.112.050 172.016.112.194 0 -
4   07/20/1998 07:59:48 00:00:01 smtp        32776 25  172.016.112.050 195.115.218.108 0 -
5   07/20/1998 08:00:01 00:00:01 domain/u    1114  53  192.168.001.010 172.016.112.020 0 -
6   07/20/1998 08:00:01 00:00:01 domain/u    1059  53  192.168.001.010 172.016.112.020 0 -
7   07/20/1998 08:00:31 00:00:01 snmp/u      1195  161 194.027.251.021 192.168.001.001 0 -
8   07/20/1998 08:00:31 00:00:01 urp/l        -     -  192.168.001.001 194.027.251.021 0 -
9   07/20/1998 08:00:36 00:00:01 snmp/u      1197  161 194.027.251.021 192.168.001.001 0 -
10  07/20/1998 08:00:41 00:00:01 snmp/u      1199  161 194.027.251.021 192.168.001.001 0 -
                                          ……
```

Figure 34. KDD’99 dataset

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 power in both time (CPU) and size (memory). For speed and effectiveness, customers' requirements need to create corresponding SESs which 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 according to 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 that is shown in Figure 33, increase efficiency and automated factors. This study examines four pragmatic frames: protocol analysis, throughput analysis, LAND attack detection, and POD attack detection. Figure 35 illustrates an ontology representing network behaviors and pragmatic frames. It shows that every frame requires different sets of attributes.

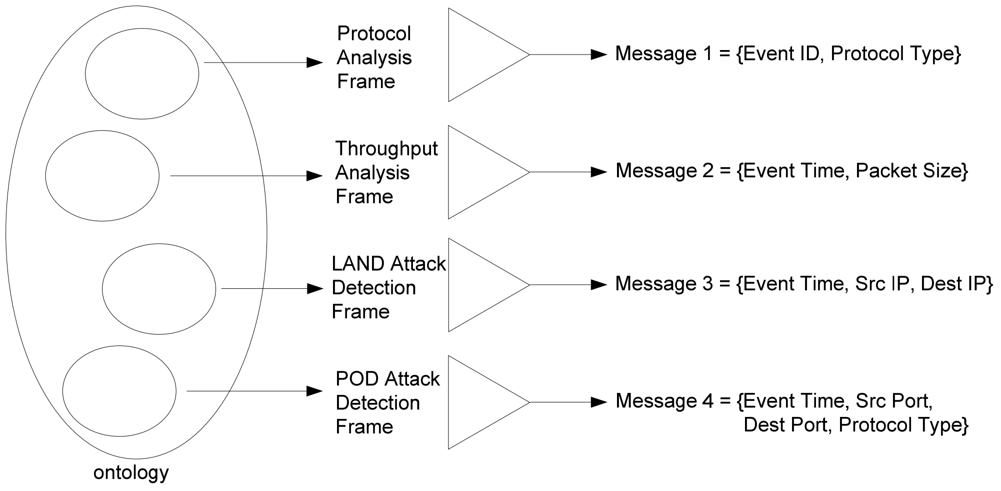

Protocol Analysis Frame
Message 1 = {Event ID, Protocol Type}
Throughput Analysis Frame
Message 2 = {Event Time, Packet Size}
LAND Attack Detection Frame
Message 3 = {Event Time, Src IP, Dest IP}
POD Attack Detection Frame
Message 4 = {Event Time, Src Port, Dest Port, Protocol Type}
ontology

Figure 35. Pragmatic frames for network traffic analysis

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 identification numbers (ID). The second analysis, measuring network throughput, needs event times and packet sizes. The two SESs, *ProtocolAnalyses* and *ThroughputAnalyses*, are described in Figure 16 and Figure 17. The third and the fourth pragmatic frames are regarding evaluating intrusion detection systems. We examine 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 computer, causing it to lock up. The LAND attack occurs when an attacker sends a spoofed SYN packet in which the source address is the same as the destination address [48]. 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 attack slows down operating speed, crashes and shuts down systems, or denies users access to services on machines. The LAND attack is recognizable because IP packets with identical source IP address and 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 destination IP address, an attribute, event time, is needed for diagnosis purposes. Figure 36 illustrates an SES for the LAND attack detection.

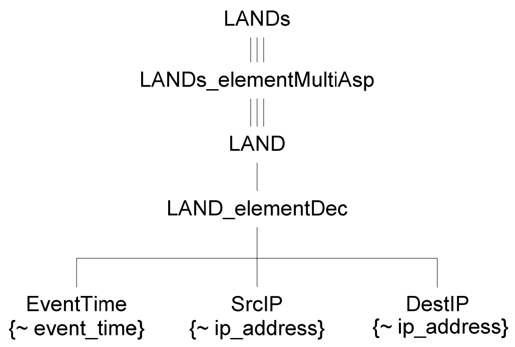


Figure 36. SES for the LAND attack detection

The Ping of Death (POD) attack is a type of Denial of Service (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 Ping of Death attack was relatively easy to carry out and very dangerous due to its high probability of success. Operating system vendors had made patches available to avoid the Ping of Death. Still, many Web sites continue to block Internet Control Message Protocol (ICMP) ping messages at their firewalls to avoid similar denial of service attacks. An attempted Ping of Death can be identified by noting the size of all ICMP packets and flagging those that are larger than 64000 bytes [48]. However, KDD'99 dataset does not have the attribute of packet size. ICMP does not have a port abstraction. 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 Ping of Death attacks with three attributes: a source host port number, a destination host port number, and a protocol. Figure 37 presents an SES for the POD attack detection.

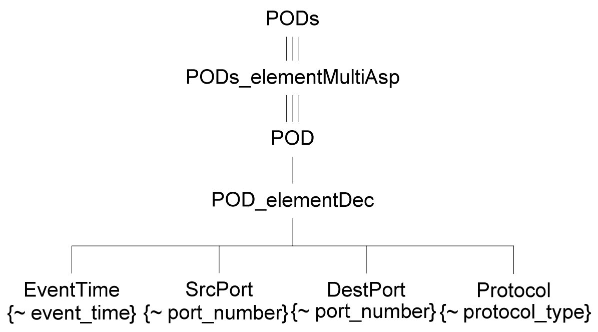


Figure 37. SES for the Ping of Death attack detection

We discuss mapping operations in Chapter 4.4.2. Mappings could be two kinds of forms: transformations and restructurings. Transformations are mappings from one representation to another and referred as general mappings. Restructurings are mappings whose domain and range are the same. That 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 an 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 pruned entity structures is defined. However, this mapping has a limitation. That 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 generic purposes of network traffic behavior analyses. New SESs are generated to correspond to customers’ requirements. But, the problem is that the structures of the two SESs, the *NetworkTrafficAnalysis* and one of the *ProtocolAnalyses*, the *ThroughputAnalysis*, 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 modeling and simulation purposes.

8.2 DEVS Service Oriented Architecture (DEVS/SOA)

DEVS simulation on Service Oriented Architecture (SOA) [50, 53] consists of three layers such as model distribution, simulation, and simulation result return. To support these layers, two services, named *MainSerivce* and *Simulation*, are implemented. *MainService* has four services, *Upload* DEVS model, *Compile* DEVS model, *Simulate* DEVS model, and *Get* result of simulation. *Simulation* service is for covering 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.

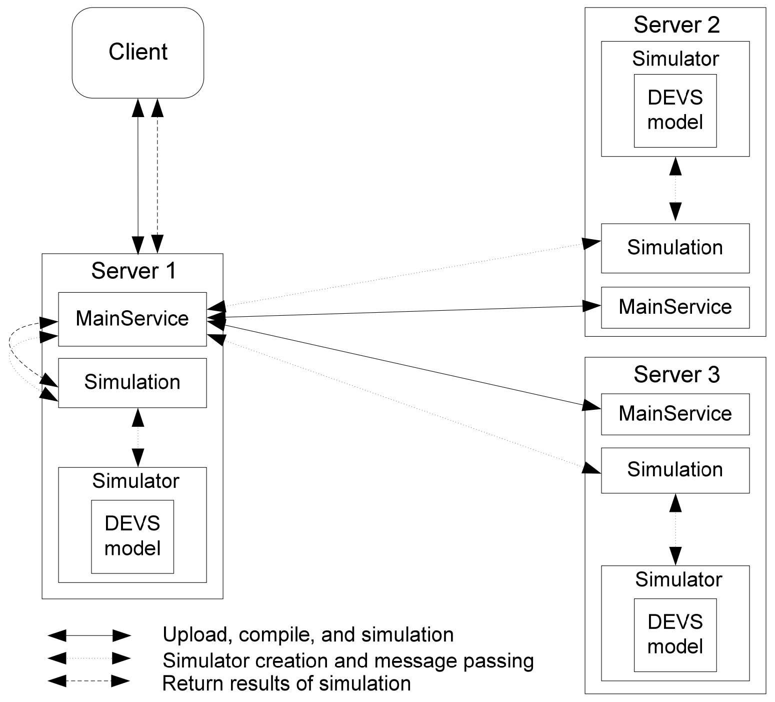


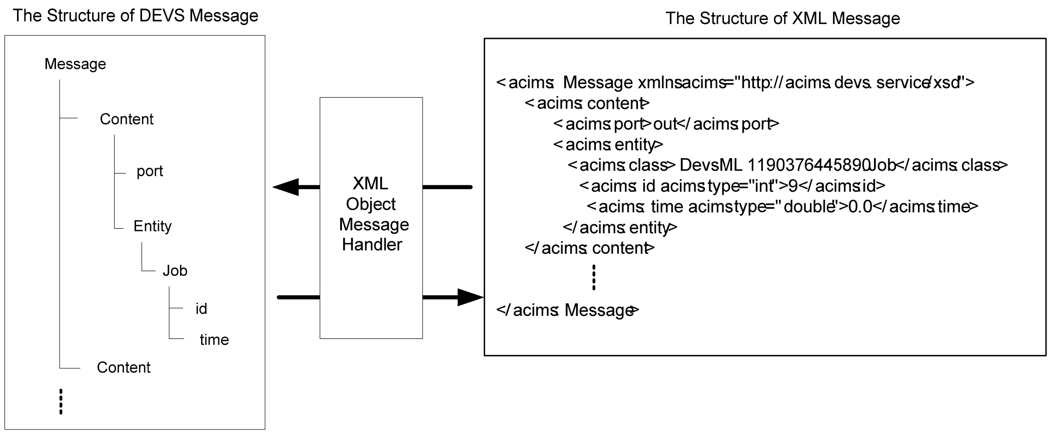
Figure 38. Overall architecture of DEVS simulation on SOA

Figure 38 represents the overall sketch of DEVS simulation on SOA. As seen in Figure 38, this system has two components, such as a client and some servers. Each server has two services (*MainService* and *Simualtion*) and the DEVS Modeling and Simulation (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 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 of 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 a 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 [50] to an atomic model with DEVS state machine because there is no support of the 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 of the simulation results to the main server. In the DEVS simulation in this study, a *Collector* DEVS atomic model collects simulation results sent from each DEVS model on each server. The main server sends the request of 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 simulation results on graphic charts. The results are stored into a file named *result.txt* processed to data format which charts use as an input.


The Structure of DEVS Message
Message
Content
port
Entity
Job
id
time
Content
XML Object Message Handler
The Structure of XML Message

```
<acims:Message xmlns:acims="http://acims.devs.service/xsd">
    <acims:content>
        <acims:port>out</acims:port>
        <acims:entity>
          <acims:class> DevsML 1190376445890Job</acims:class>
            <acims:id acims:type="int">9</acims:id>
              <acims:time acims:type="double">0.0</acims:time>
          </acims:entity>
    </acims:content>
            ⋮
</acims:Message>
```

Figure 39. Example of XML object message handler

Models upload is done through serialization and SOA technologies, and message passing is done through XML style message and SOA technologies. Figure 39 is an example of a DEVS message to a XML-style message conversion. A DEVS message is a language specific object class, and Web Service does not have an apparatus to send an arbitrary object message to another service because Web Service supports only fixed structured messages defined in WSDL. A DEVS message is too dynamic to define it as one type of classes in the WSDL. So, XML Object message handler is employed to transform an object DEVS message to a XML-style message. As seen in Figure 39, the structure of DEVS message consists of at least more than one contents containing a port and Entity object. Entity objects can be any type of objects inherited by Entity. This DEVS message is converted to a 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's 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 a XML-style message produced by XML Object message handler. After the message sending is finished, 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 above procedures until simulation termination condition meets.

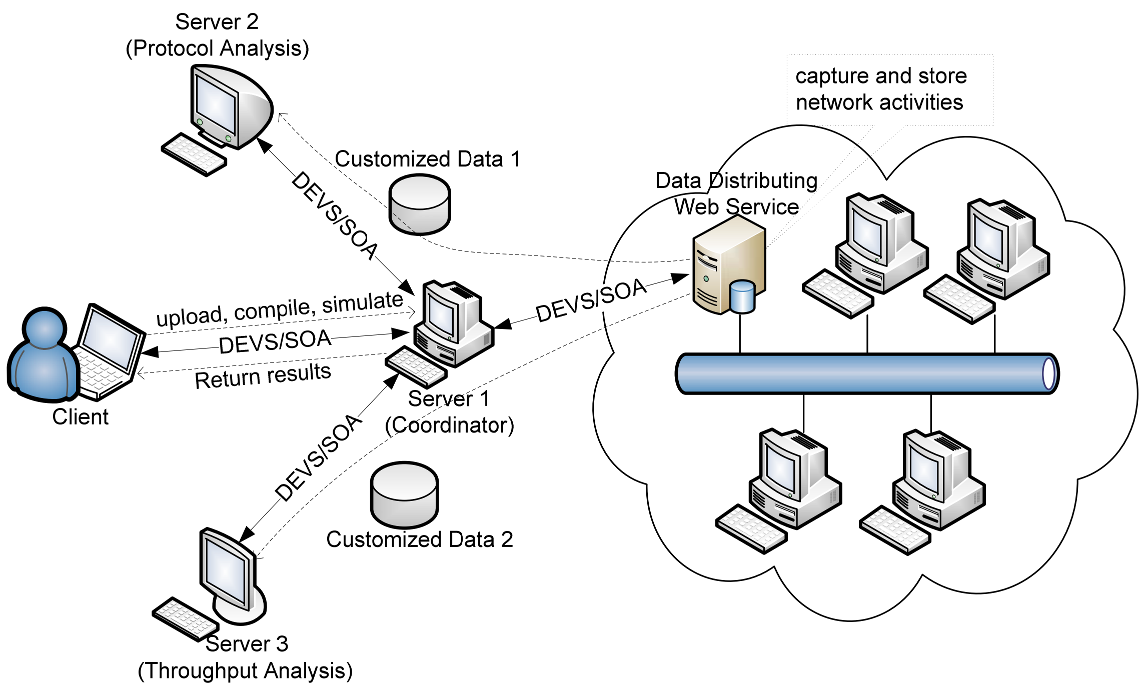


Figure 40. A network behavior analysis using DEVS/SOA

Figure 40 illustrates a DEVS simulation on SOA which is applied to a network behavior analysis example which is the case that 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: a server 1 for acting as a coordinator, a server 2 for analyzing protocol uses, and a server 3 for measuring network throughput, out of the subnet. The four servers (one in 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 the server 2 and throughput analysis to the 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 time (CPU) and space (memory). These two factors, distributed workloads and small size customized data, enable clients to obtain simulation results fast and efficiently.

8.2.1 Real Time DEVS Simulation on SOA

Other approach of DEVS Simulation on SOA is real time simulation in which next time for occurring internal transition passes by real time. Unlike virtual time simulation, time synchronizes simulation protocol to simulate DEVS models on SOA, and real time DEVS simulation has minimum network activity among simulators because the simulators only invoke web services at the time of the propagation of out messages. Also it is decentralized simulation because there is no coordinator to supervise all RTSimulators. Each RTSsimulator follows a procedure to simulate their DEVS model without intervention for synchronization.

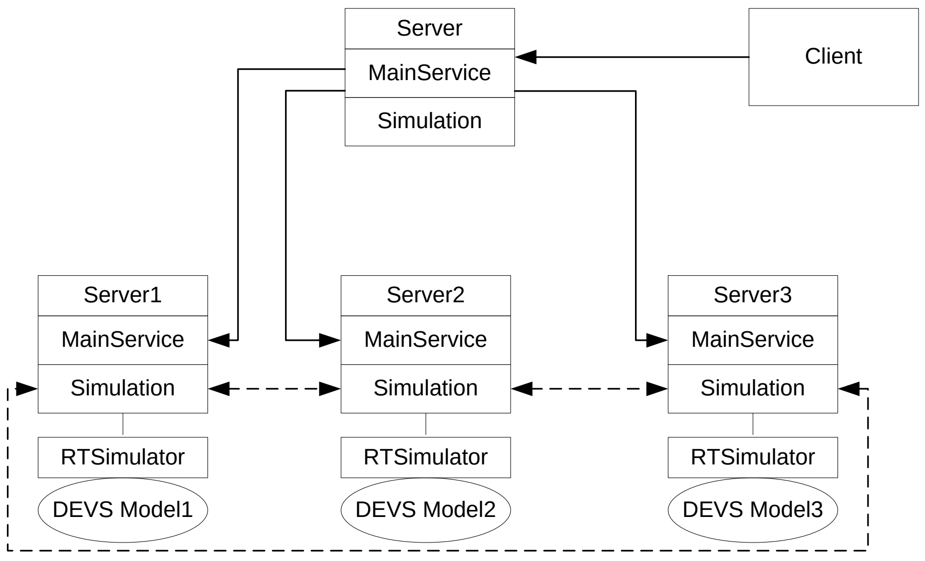


Figure 41. Overall architecture of real time DEVS simulation system on SOA

Figure 41 represents overall structure of real time DEVS simulation system on SOA. As seen in the figure 41, each server participating in simulation has two web services similar to centralized simulation. But some functions in the simulation service and classes such as *RTCoordinator* and *RTSimulator*, are added to support real time simulation. *RTCoordinator* used in the *MainService* and *RTSimulator* used in the *Simualtion* are made of multi-threads. *RTCoordinator* generates proxies for *Simulation* services with DEVS models and coupling information which 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 like centralized simulation on SOA. Solid-lines on figure 41 represent uploading files, compiling the files on each server, and executing *RTSimulators* on *Simulation* services. Dashed-lines show out message passing routes.

Figure 42 depicts real time DEVS simulation protocol. The protocol starts with the initialization of the DEVS models in the *RTSimulators*. Each *RTSimulator* waits for passing *tN* after which internal transition occurs. If one of the *RTSimulators* has wall-clock time equal to *tN*, the *RTSimulator* executes internal transition function consisting of *lamda* function which produces an out message, propagation function which sends the out message to other *RTSimulators* according to coupling information, and delta function which handles internal and external events. *RTSimulator2* in the figure 42 shows "send out message" after internal transition and wait again with *tN* regenerated by *delta* function. Meanwhile, *RTSimulator1* receives a message from the *RTSimulator2*, executes external transition function having delta function, and recalculates *tN* to wait. The interaction between *RTSimulator2* and *RTSimulator1* does not affect *RTSimulator3*. Only way to influence to others send a message to others.

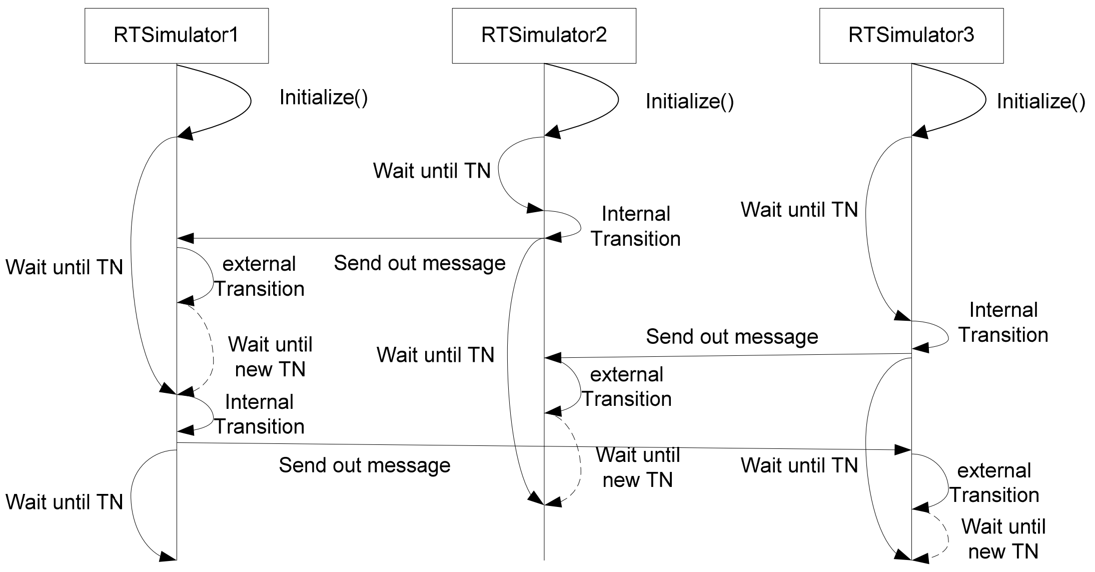


Figure 42. Real time simulation protocol

Though real time DEVS simulation has minimum network traffic, in case of network delay and tiny value of *tN*, the simulation might fail to get correct results because of distorted protocol. To filter the problem, it is important to know threshold value of *tN* to make real time DEVS simulation done or speed up.

8.3 Distributed SES/NZER in DEVS/SOA

8.3.1 Distributed Simulation

A distributed SES/NZER is different to classic single machine DEVS simulation. In this chapter, 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 43 shows distributed DEVS simulation.

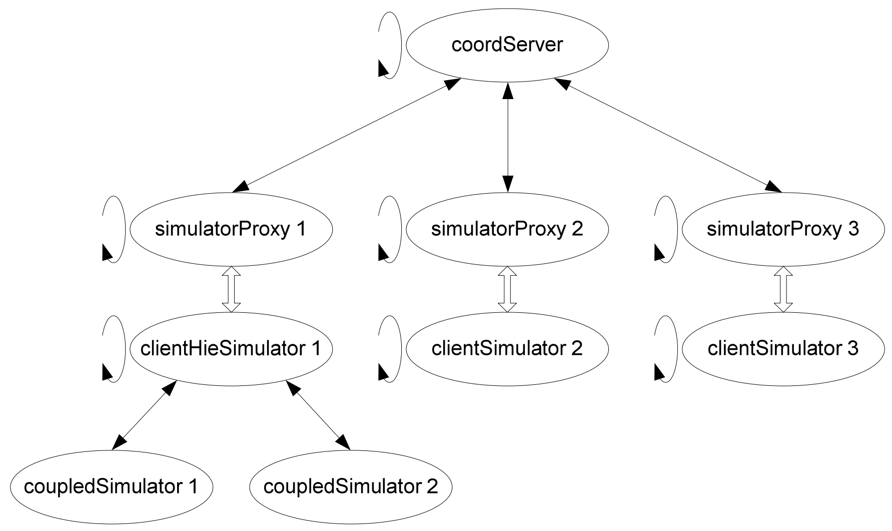


Figure 43. Distributed DEVS simulation

For distributed DEVS simulation, there must be a controller, a *coordServer*, which manages a whole simulation cycle and synchronizes all the distributed simulators. The *coordServer* is responsible for passing messages among distributed simulators as well as for advancing DEVS models which are dispersed in networks. The *coordServer* could be in a host which 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 *coupledSimualtor* to take care of a local DEVS atomic model. The *coordServer* creates *simulatorProxys* that facilitate the *coordServer* communicating with corresponding *clientSimulators* or *clientHieSimulators*. In addition, all the distributed components, the *coordServer*, the *simulatorProxys*, the *clientSimulators*, and the *clientHieSimulators*, has its own thread. Figure 44 shows an example of DEVS modeling for a distributed simulation for network traffic analysis (SimForNTA).

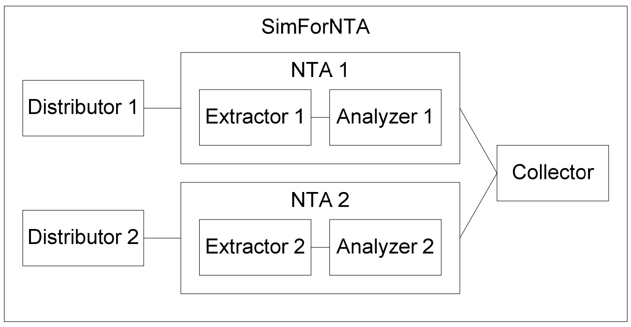


Figure 44. DEVS modeling: Distributed SimForNTA

The top level of coupled model is a *SimForNTA*. The *SimForNTA* is composed of two coupled models, a *NTA 1* and a *NTA 2*, and three atomic models, a *Distribute 1*, a *Distributor 2*, and a *Collector*. Two sub coupled models (the *NTA 1* and the *NTA 2*) include their own components (a *Extractor* and a *Analyzer*). To achieve fast analysis time, we apply the divide and conquer approach. A 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 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 analyzed results from the two processes. We assign all the models to different computers which are connected in networks. Figure 45 illustrates a hierarchically structured distributed DEVS simulator and corresponding DEVS models.

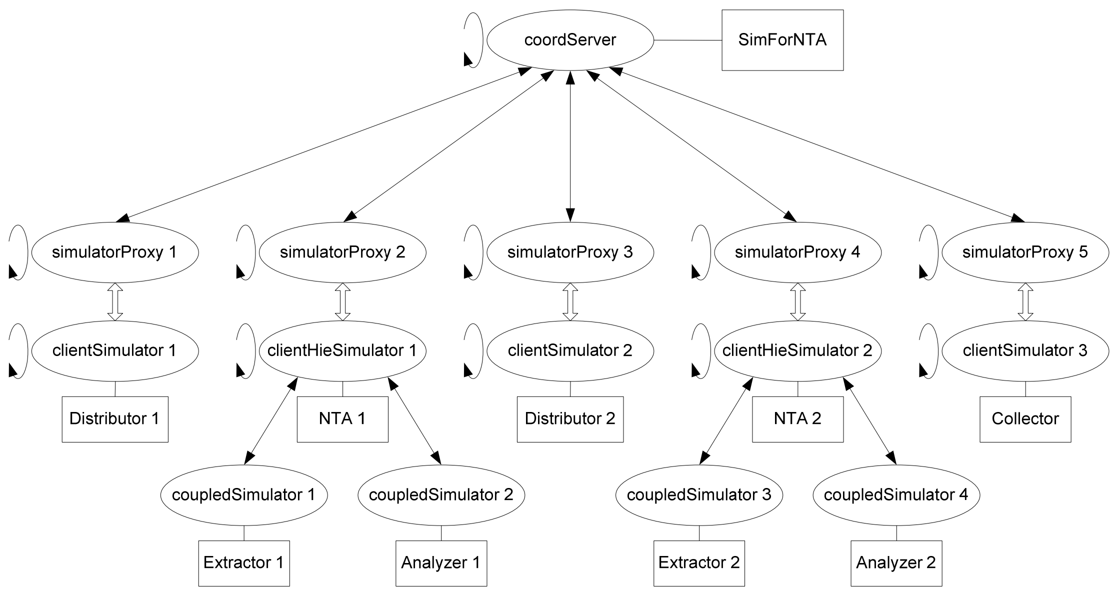


Figure 45. Distributed DEVS simulators and models for SimForNTA

In this example, the top level coupled model, the *SimForNTA*, two sub coupled models, the *NTA 1* and the *NTA 2*, and three atomic models, the *Distributor 1*, the *Distributor 2*, and the 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 *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 a *coordServer*, *simulatorProxys*, *clientSimulators*, and *clientHieSimulator* have their own thread, the slowest thread determines the overall simulation speed in the divide and conquer mechanism because the divide and conquer is a pipeline with divider, processors (in parallel) and compiler so the slowest one of these stages determines the overall speed. Therefore, speeding up all the threads is important. And, reducing communication overhead over networks is also a critical issue in distributed simulation environments.

8.3.2 DEVS Modeling and Simulation

Although a distributed SES/NZER follows decentralized distributed DEVS simulation scheme, couplings among components (DEVS atomic models and DEVS coupled models) keep the function as single machine DEVS. For example, a coupled model, *coupledModel*, is composed of three atomic models: *atomicModel 1*, *atomicModel 2*, and *atomicModel 3*, we could assign the *atomicModel 1* to a host 1, the *atomicModel 2* to a host 2, the *atomicModel 3* to a host 3, and the *coupledModel* to a host 4 or one of the hosts which hold the atomic models. Therefore, the *coupledModel* controls synchronization among the atomic 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, Discrete Event Specification (DEVS) is used for simulation engine. To advance simulation cycle, 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 Figure 46:

1. Coordinator sends a *nextTN* message to request next event time (*tN*) from each of the simulators.

2. All the simulators reply with their tNs in an *outTN* message back to the coordinator
3. Coordinator sends to each simulator a *getOut* message containing the global *tN* (the minimum of the *tNs*)
4. Each simulator checks if it is imminent, which means its *tN* equals to global *tN*, and if so, returns an output of its model in a message to the coordinator in a *sendOut* message.
5. 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.

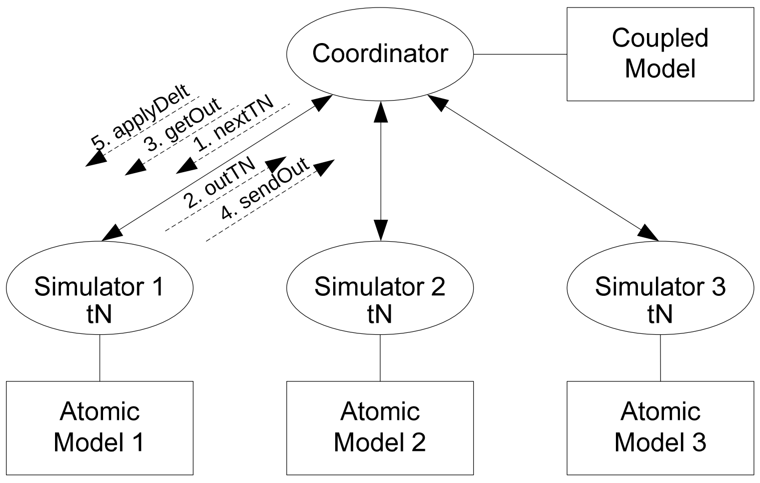


Figure 46. Basic DEVS simulation protocol

The basic DEVS simulation protocol is illustrated in Figure 46. If a coupled model and all the atomic models are assigned in different machines which are connected in networks, DEVS protocol overheads may exceed the advantage of distributed simulation deploying workloads. Diminishing the number of DEVS protocol messages among computers results in decreasing communication overheads. Therefore, we may expect 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 networks of systems as components in a larger coupled system, leading to hierarchical, modular construction [17]. This means that every coupled model is behaviorally equivalent to a basic atomic model.

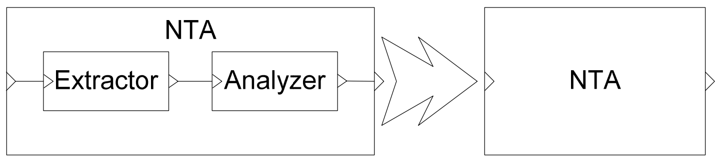


Figure 47. Closure under coupling for SimForNTA

Figure 47 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 to be one component, the *NTA* atomic model. We translate the coupling information of coupled model, *NTA*, into a flat-structured atomic model, *NTA*. By this translation, hierarchical structure of the DEVS model can be flattened. Message exchanges consume a large amount of time if the model structure is too complex or extremely large in distributed environments. If the model hierarchy is flattened, communication overheads among models can be minimized. Therefore, flat-structured modeling approach facilitates to reduce the number of messages, and we can achieve better performance results [58, 59]. 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 *simulatorProxy* for a message. If the atomic model NTA replaces the three component DEVS model, only one *simulatorProxy* is created by the *coorServer*. As a result, we could obtain speed up. Figure 48 shows that the closure under coupling decreases the number of *simulatorProxys* and simplifies the DEVS simulation architecture. The left diagram illustrates a simulation environment before DEVS models are refined, and the right figure presents the refined DEVS model.

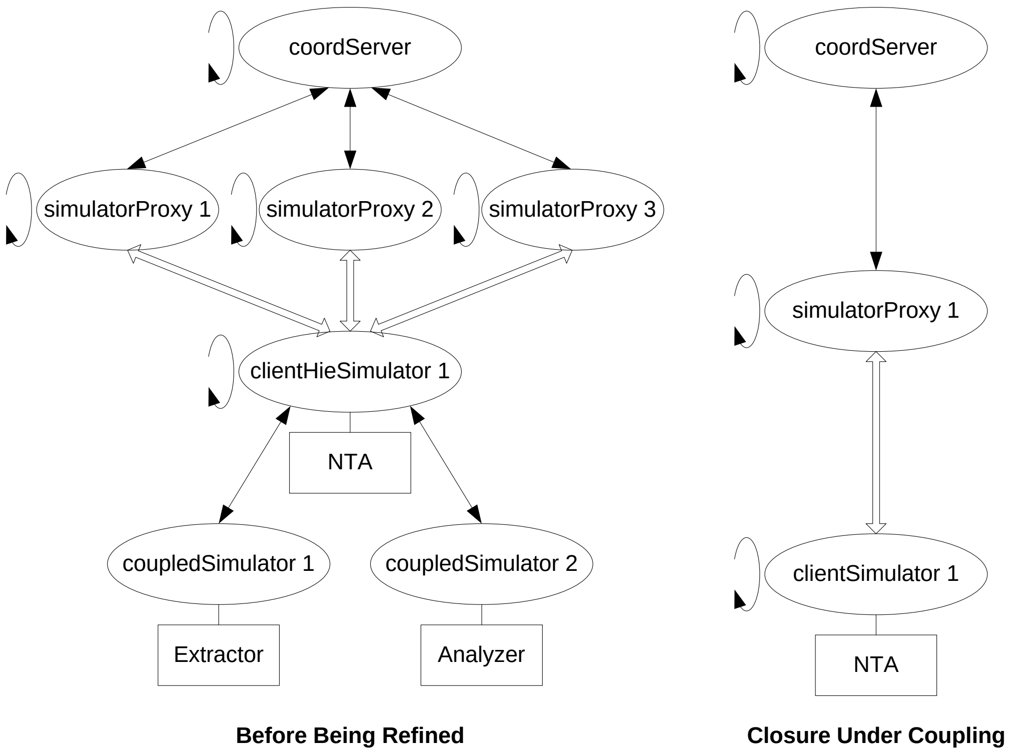


Figure 48. DEVS model comparison under the DEVS/SOA environment

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*. Processing time for these DEVS protocol messages transmissions should not overwhelm processing time of the processor. An atomic model of SES/NZER loads PES XML documents and analyzes one tuple information at one state transition. This approach needs many state transitions according to the number of tuples in PEX XML files. For example, there are ten PES XML files, and each PES XML file includes 1,300 tuples 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 PES XML documents, 1300*10 iterative transitions (the *analyze* state to the *analyze* state) for evaluating all the tuples in ten PESs, and 10 transitions (the *analyze* state to the *extract* state) to load PES files. Figure 49 shows these state transitions.

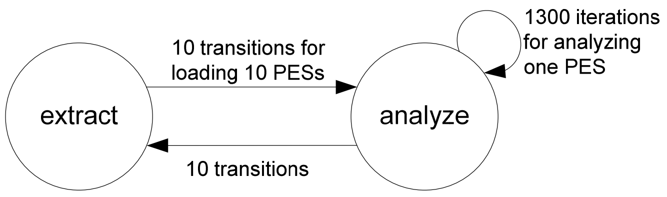

extract
analyze
10 transitions for loading 10 PESs
10 transitions
1300 iterations for analyzing one PES

Figure 49. State diagram in SES/NZER

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, there is too much communication overhead for only advancing simulation cycle. So, we fit SES/NZER's atomic models to a distributed simulation. An *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 50 illustrates an updated state transition diagram for the distributed SES/NZER.

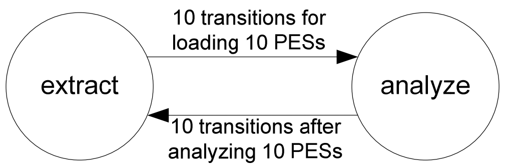

10 transitions for loading 10 PESs
extract
analyze
10 transitions after analyzing 10 PESs

Figure 50. State diagram in 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 overall simulation time over network environments. The Lee's Ph.D dissertation [60] discusses the effect of quantization in distributed DEVS/HLA environments. Communication latency and overhead reduction technique in distributed interactive simulation are introduced through an approach of bundling Protocol Data Unit (PDU) [61].

CHAPTER 9. EXPERIMENTAL RESULTS

We set up a testbed for a distributed simulation environment in the ACIMS lab as shown in Figure 51. We install 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 [62]. We install an Apache Axis2/Java Web service engine [63]. Apache Axis2 is the core engine for Web services, and it is an implementation of the World Wide Web Consortium (W3C) Simple Object Access Protocol (SOAP). W3C defines SOAP as below:

> *SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework. [64]*

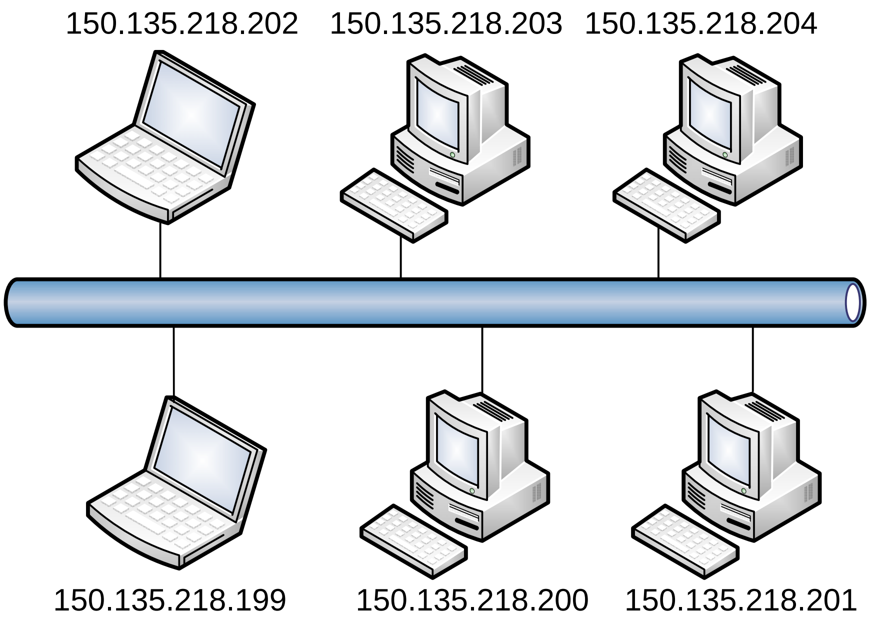
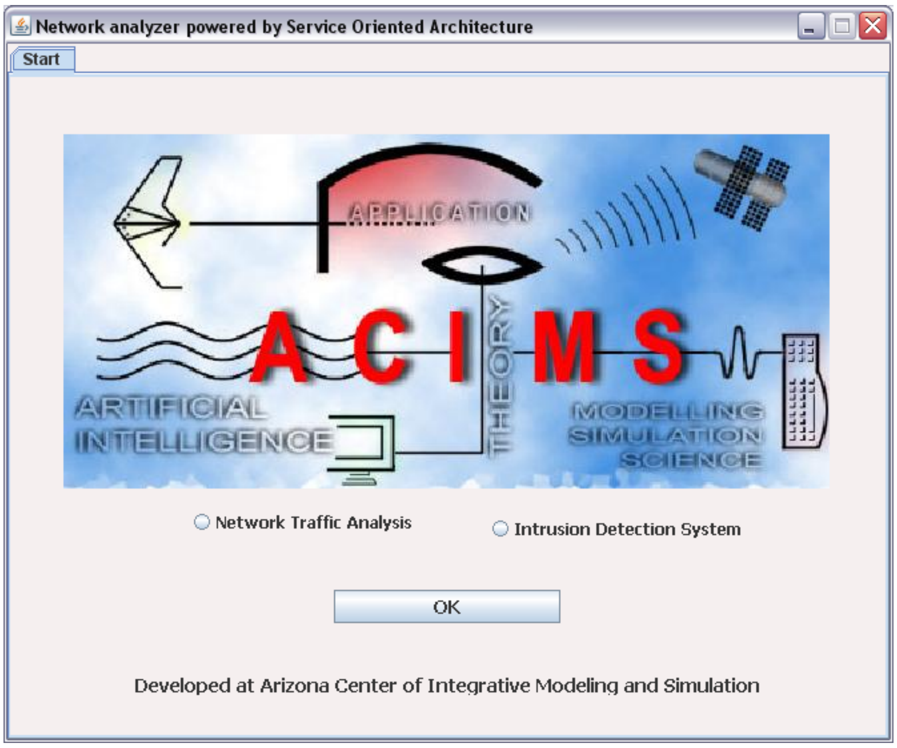


Figure 51. Testbed for distributed simulation using DEVS/SOA

We monitor and capture network activities inside the ACIMS lab subnet, and we use the captured data for generic network behavior analyses such as protocol evaluation and throughput measurement. For intrusion detection analyses, the KDD'99 dataset is used as source data because the KDD'99 dataset was originally generated for the purpose of intrusion detection system researches, and the dataset includes various attacks as well as normal packet transmission events. Figure 52 shows the main GUI of the distributed SES/NZER. There are two functions: network traffic analysis and intrusion detection system.

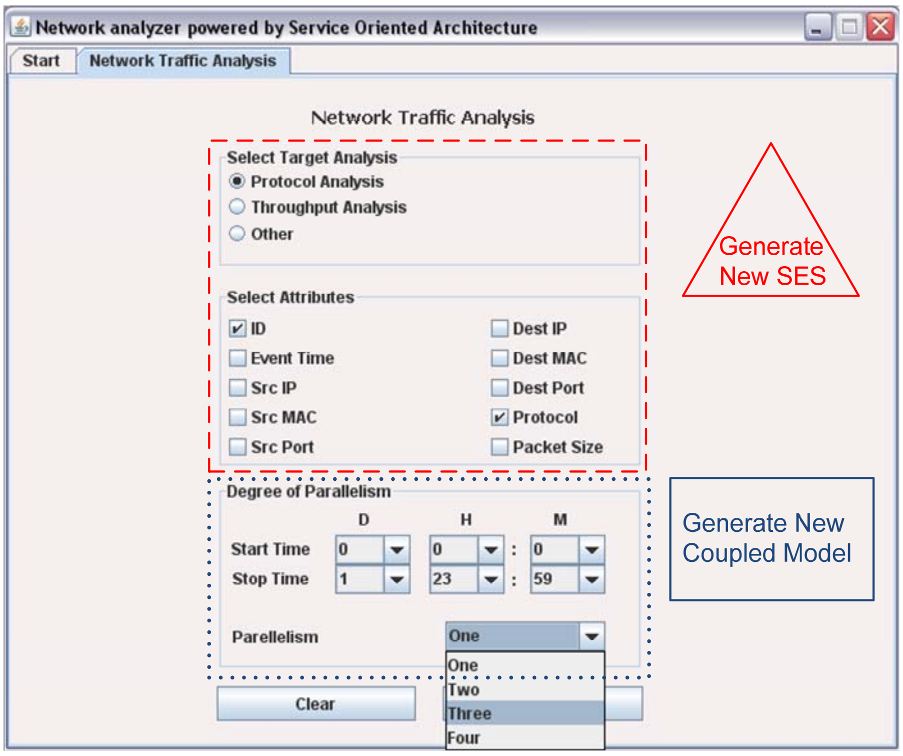

Network analyzer powered by Service Oriented Architecture
Start
APPLICATION
ACIMS
THEORY
ARTIFICIAL INTELLIGENCE
MODELLING SIMULATION SCIENCE
Network Traffic Analysis
Intrusion Detection System
OK

Developed at Arizona Center of Integrative Modeling and Simulation

Figure 52. Main GUI of distributed SES/NZER

9.1 Network Traffic Analysis

This chapter 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 which is shown in Figure 53. According to target analyses, corresponding required attributes are selected automatically. Or, users could choose attributes if they want 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. Then, deciding time frames is next. Finally, 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 model partitioning approach in distributed simulation is proposed and implemented in the Zhang's Ph.D dissertation [65].


Network analyzer powered by Service Oriented Architecture

Start | Network Traffic Analysis

Network Traffic Analysis

Select Target Analysis
- [●] Protocol Analysis
- [ ] Throughput Analysis
- [ ] Other

Select Attributes
- [✓] ID
- [ ] Event Time
- [ ] Src IP
- [ ] Src MAC
- [ ] Src Port
- [ ] Dest IP
- [ ] Dest MAC
- [ ] Dest Port
- [✓] Protocol
- [ ] Packet Size

Degree of Parallelism

	D	H	M
Start Time	0	0	0
Stop Time	1	23	59

Parellelism: One

Figure 53. User request input system for network traffic analyses

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 54 shows the processes of choosing a top level coupled model and assigning models into distributed servers.

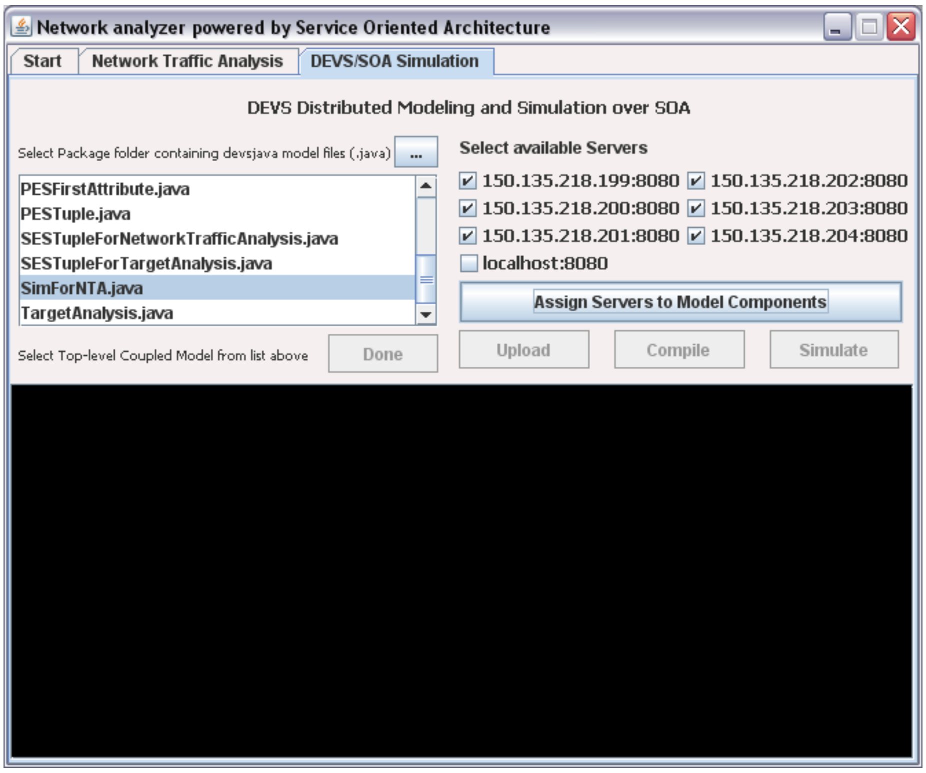


Figure 54. Assign models into multiple servers

We measure the data size, and the original data sizes for half-day, one day, and two days are 2.83 GB, 5.44 GB, and 10.8 GB. Instead of keeping all the attributes, PES XML documents for protocol analysis holds two attributes: a packet ID and a protocol type. So, the PES file sizes are 168 MB, 326 MB, and 646 MB. Their sizes are about six percent of the original data size. PES files for throughput evaluation include two attributes, an event time and a packet size, and their sizes are 200MB, 387MB, and 770

MB. The ratio is about seven percent. Table 8 presents the data size comparisons between original data and PES data for network traffic analyses.

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

Table 8. Data size comparisons for network traffic analyses

In addition to measuring the data size, we examine execution times of half-day, one day, and two days data of both protocol analysis and throughput measurement by varying degree of parallelism (numbers of computer for analysis). We experiment 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 is composed of three sub-times: time for distributing data to servers, time for evaluating received data at analyzing servers, and time for collecting and displaying evaluated results at a client computer. Table 9 shows the execution times of protocol analyses, and Table 10 presents the simulation times of throughput analyses in virtual time DEVS/SOA.

Data	Execution Times	Local	1*1	2*2	3*3

Half day	Distributing time	8min 23sec	8min 18sec	12min 28sec	10min 5sec
	Analyzing time	1min 26sec	1min 12sec	57sec	59sec
	Collecting time	1sec	2sec	1sec	1sec
	Total	**8min 50sec**	**9min 32 sec**	**13min 26sec**	**11min 5sec**
One day	Distributing time	16min 15sec	15min 23sec	18min 16sec	16min 29sec
	Analyzing time	2min 20sec	2min 1sec	1min 47sec	1min 59sec
	Collecting time	1sec	1sec	1sec	1sec
	Total	**18min 35sec**	**17min 25sec**	**20min 4sec**	**18min 29sec**
Two days	Distributing time	30min 34sec	32min 57sec	35min 47sec	33min 21sec
	Analyzing time	4min 43sec	4min 34sec	3min 32sec	3min 56sec
	Collecting time	1sec	1sec	1sec	2sec
	Total	**35min 18sec**	**37min 32sec**	**39min 20sec**	**37min 19sec**

Table 9. Execution times of protocol analyses in virtual time simulation

Data	Execution Times	Local	1*1	2*2	3*3
Half day	Distributing time	9min 26sec	10min 13sec	14min 15sec	12min 13sec
	Analyzing time	1min 8sec	1min 4sec	57sec	1min 3sec
	Collecting time	1sec	1sec	1sec	1sec
	Total	**10min 35sec**	**11min 18sec**	**15min 13sec**	**13min 27sec**
One day	Distributing time	18min 13sec	19min 37sec	22min 11sec	21min 46sec
	Analyzing time	2min 18sec	2min 2sec	1min 51sec	2min 13sec

	Collecting time	1sec	1sec	1sec	1sec
	Total	**20min 32sec**	**21min 40sec**	**24min 3sec**	**24min 0sec**
Two days	Distributing time	34min 1sec	39min 58sec	39min 37sec	42min 14sec
	Analyzing time	4min 0sec	5min 53sec	5min 26sec	3min 58sec
	Collecting time	1sec	1sec	1sec	1sec
	Total	**38min 2sec**	**45min 52sec**	**45min 4sec**	**46min 13sec**

Table 10. Execution times of throughput analyses in virtual time simulation

For three different datasets, we measure three kinds of times: distributing data time, analyzing data time, and collecting resulting data time. We measure execution times at four different sets of computers: {a local computer}, {one distributing data computer, one analyzing data computer}, {two distributing data computers, two analyzing data computers}, and {three distributing data computers, three analyzing data computers}. We notice that distributing times increase gradually as the number of distributed computers increases. Ideally, distributing times must decrease in the counter ratio 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 getting smaller as the number of computers is getting larger. Against distributing times, analyzing times are not affected by network communication overheads. Because collecting resulting data times are one second or two seconds in most cases, we could forgo collecting times for comparing execution times. Figure 55 illustrates virtual time simulation results of network behavior analyses.

Virtual Time Protocol Analysis	Virtual Time Throughput Analysis

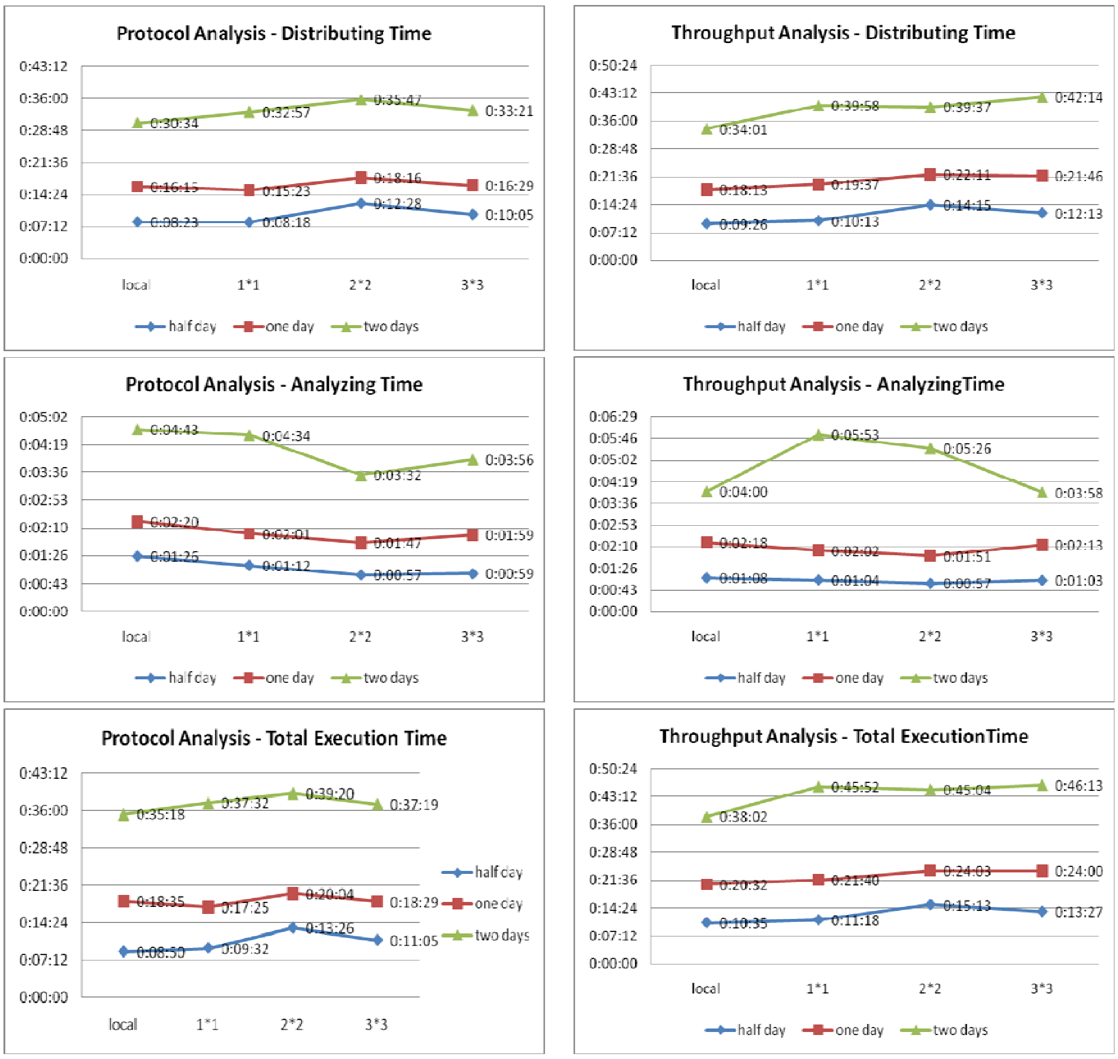


Figure 55. Virtual time simulation results of network behavior analyses

In virtual time DEVS/SOA simulation, all 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 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, real time DEVS/SOA simulation is applied, and, finally, we accomplish a goal of distributed simulation, speed up execution times, through real time simulation. Because each simulator in different machine has own simulation time, and overall execution time is not affected by communication overheads which are caused by DEVS protocol messages and data messages between a centralized coordinator and distributed simulators, we achieve speed up comparing to virtual time simulation. Table 11 shows the execution times of protocol analyses, and Table 12 presents the simulation times of throughput analyses in real time DEVS/SOA. Figure 56 illustrates real time simulation results for both protocol and throughput analyses.

Data	Execution Times	Local	1*1	2*2	3*3
Half day	Distributing time	6min 10sec	6min 53sec	6min 1sec	2min 20sec
	Analyzing time	1min 9sec	30sec	15sec	9sec
	Collecting time	1sec	1sec	1sec	1sec
	Total	**7min 20sec**	**7min 24sec**	**6min 17sec**	**2min 30sec**
One day	Distributing time	11min 55sec	12min 24sec	7min 0sec	3min 40sec
	Analyzing time	2min 16sec	53sec	21sec	16sec

	Collecting time	1sec	1sec	1Sec	1sec
	Total	**14min 12sec**	**13min 18sec**	**7min 22sec**	**3min 57sec**
Two days	Distributing time	26min 36sec	27min 18sec	12min 2sec	7min 37sec
	Analyzing time	3min 47sec	3min 44sec	54sec	36sec
	Collecting time	1sec	1sec	1sec	1sec
	Total	**30min 24sec**	**31min 3sec**	**12min 57sec**	**8min 14sec**

Table 11. Execution times of protocol analyses in real time simulation

Data	Execution Times	Local	1*1	2*2	3*3
Half day	Distributing time	7min 29sec	8min 25sec	7min 00sec	3min 24sec
	Analyzing time	1min 23sec	29sec	15sec	10sec
	Collecting time	1sec	1sec	1sec	1sec
	Total	**8min 53sec**	**8min 55sec**	**7min 16sec**	**3min 35sec**
One day	Distributing time	15min 15sec	15min 24sec	8min 57sec	4min 54sec
	Analyzing time	2min 24sec	1min 35sec	18sec	12sec
	Collecting time	1sec	1sec	1sec	1sec
	Total	**17min 40sec**	**17min 0sec**	**9min 16sec**	**5min 7sec**
Two days	Distributing time	30min 22sec	33min 27sec	16min 7sec	10min 4sec
	Analyzing time	3min 58sec	3min 37sec	1min 32sec	37sec
	Collecting time	1sec	1sec	1sec	1sec
	Total	**34min 21sec**	**37min 5sec**	**17min 40sec**	**10min 42sec**

Table 12. Execution times of throughput analyses in real time simulation

Real Time Protocol Analysis Real Time Throughput Analysis

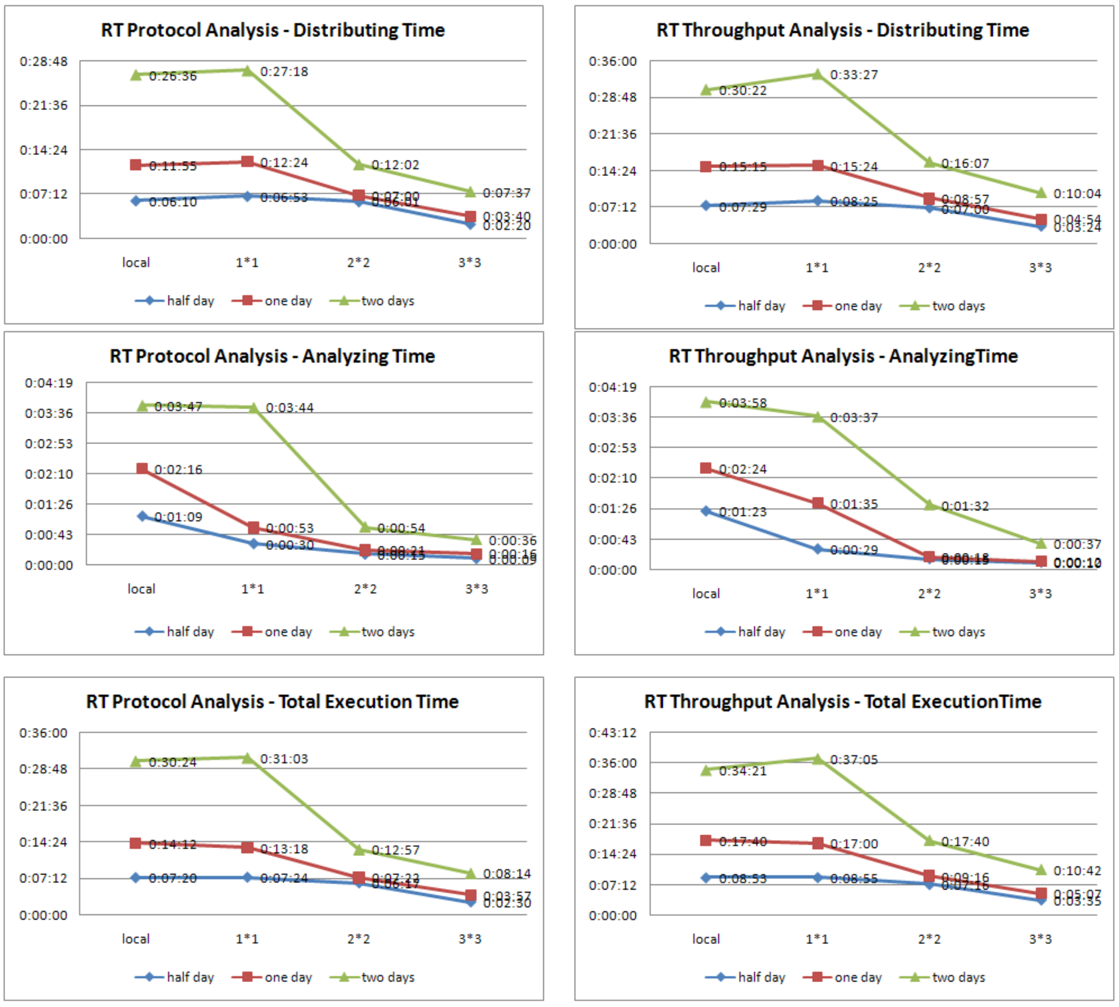
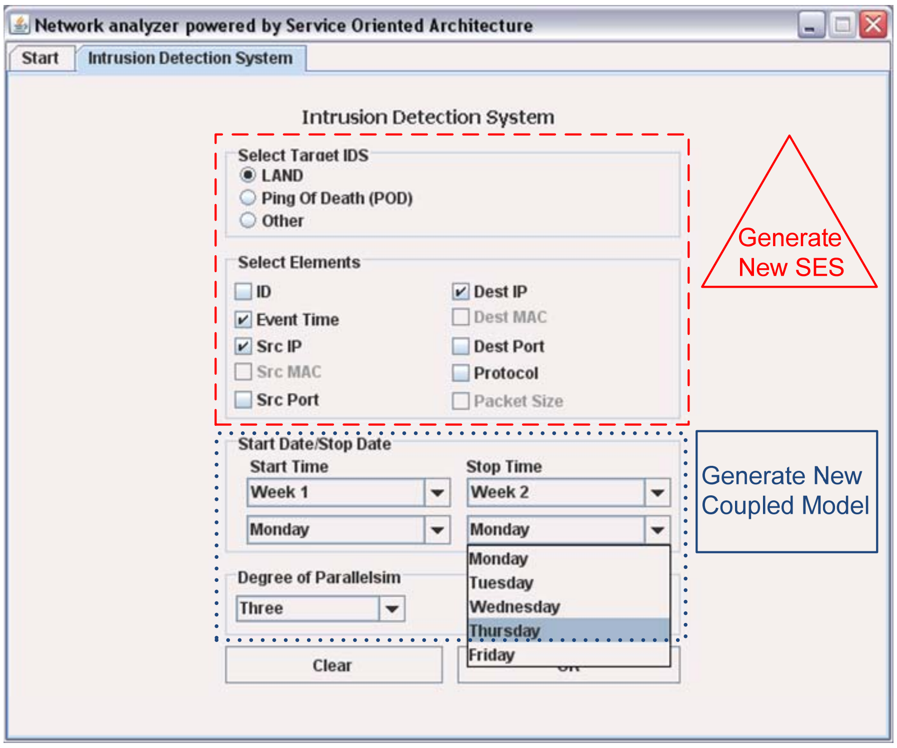


Figure 56. Real time simulation results of network behavior analyses

9.2 Intrusion Detection System

Recall that we built two IDS agent models: the LAND agent and the POD agent. As illustrated in Chapter 8.1., after customers' requests, which are selecting a target IDS, time frames (start time and end time) and a degree of parallelism (the number of distributed computer for analysis) are applied through an input system, users could assign simulation models into multiple servers according to the selected degree of parallelism. Figure 57 shows the user's request input system for evaluating intrusion detection systems. Users' requests generated both new SESs and new DEVS coupled models for evaluating IDSs.


Network analyzer powered by Service Oriented Architecture

Start | Intrusion Detection System

Intrusion Detection System

Select Target IDS: [✓ LAND] Ping Of Death (POD) Other

Select Elements:

ID	[✓ Dest IP]
[✓ Event Time]	Dest MAC
[✓ Src IP]	Dest Port
Src MAC	Protocol
Src Port	Packet Size

Start Date/Stop Date

Start Time	Stop Time
Week 1	Week 2
Monday	Monday

Degree of Parallelsim: Three

Figure 57. User request input system for intrusion detection systems

First, we measure the data sizes. The original source data size for two weeks (five days a week) is 4.12 GB. The pruned data size for LAND IDS, which include even times, source host IP addresses, and destination host IP addresses size, is 368 MB. The data size for POD IDS is 437 MB. These PES data sizes are about 9 percent (LAND) and 10 percent (POD) of the original KDD'99 dataset. Table 13 presents the data size comparisons for IDS evaluations.

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

Table 13. Data size comparisons for IDS evaluations

Also, we observe IDS evaluating times of both the LAND attack and the POD attack using the two weeks of the KDD'99 dataset. We differentiate the number of computers like we experiment for generic network traffic analysis. We achieve similar execution times to what we gain in Chapter 8.1. We notice that distributing data times are getting larger as the number of evaluating machines increases due to overheads (network packet transmission delays and DEVS protocol message overheads). We speed up analyzing times.

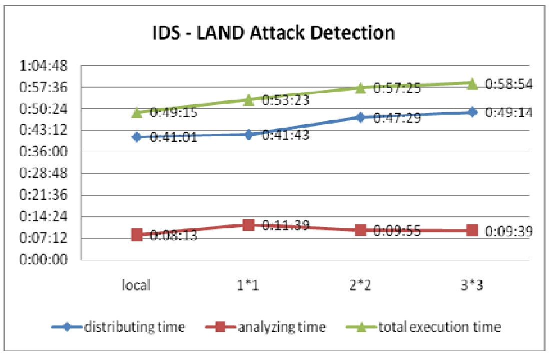
	Local	1*1	2*2	3*3
Distributing time	41min 1sec	41min 43sec	47min 29sec	49min 14sec
Analyzing time	8min 13sec	11min 39sec	9min 55sec	9min 39sec
Collecting time	1sec	1sec	1sec	1sec
Total	**49min 15sec**	**53min 23sec**	**57min 25sec**	**58min 54sec**

Table 14. Execution times of LAND attack detection in virtual time simulation

	Local	1*1	2*2	3*3
Distributing time	32min 36sec	36min 1sec	38min 33sec	39min 24sec
Analyzing time	8min 3sec	12min 1sec	10min 1sec	9min 11sec
Collecting time	1sec	1sec	1sec	1sec
Total	**40min 40sec**	**48min 3sec**	**48min 35sec**	**48min 36sec**

Table 15. Execution times of POD attack detection in virtual time simulation

Virtual Time IDS LAND Attack Detection



Virtual Time IDS POD Attack Detection

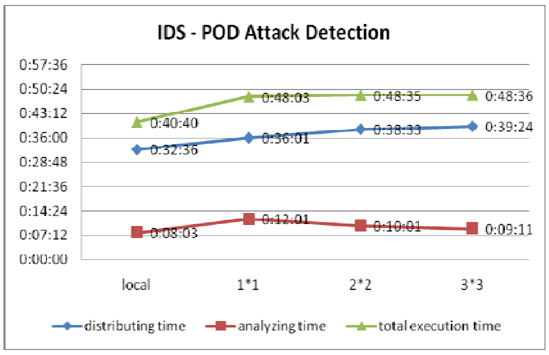


Figure 58. Virtual time experimental results of IDS analyses

Table 14, Table 15, and Figure 58 illustrate that there is no big improvement in total execution times (adding distributing time, analyzing time, and collecting time) in virtual time simulation environment. A distributed SES/ZER speeds up analyzing times by dividing a whole workload into several small jobs and deploying the small works into multiple machines. Experimental results in real time simulation are presented in table 16, table 17, and figure 59.

	Local	1*1	2*2	3*3
Distributing time	48min 50sec	41min 7sec	22min 58sec	16min 3sec
Analyzing time	11min 57sec	12min 55sec	2min 54sec	2min 8sec
Collecting time	1sec	1sec	1sec	1sec
Total	**1hr 0min 48sec**	**54min 3sec**	**25min 53sec**	**18min 14sec**

Table 16. Execution times of LAND attack detection in real time simulation

	Local	1*1	2*2	3*3
Distributing time	44min 43sec	35min 58sec	18min 21sec	11min 59sec
Analyzing time	12min 43sec	8min 38sec	5min 39sec	3min 25sec
Collecting time	1sec	1sec	1sec	1sec
Total	**57min 27sec**	**44min 37sec**	**24min 1sec**	**15min 25sec**

Table 17. Execution times of POD attack detection in real time simulation

Real Time IDS LAND Attack Detection Real Time IDS POD Attack Detection

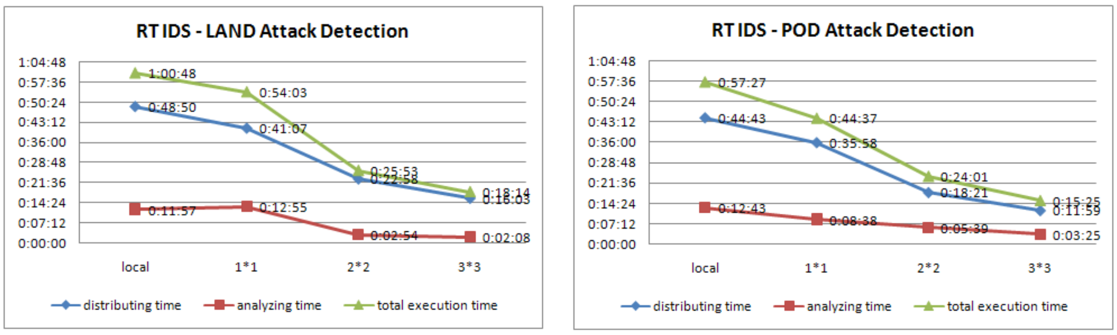


Figure 59. Real time experimental results of IDS analyses

We accomplish speed up of total execution times in 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. And, a distributed SES/ZER speeds up analyzing times by dividing a whole workload into several small jobs and deploying the small works into multiple machines. In addition, we achieve fast execution times in real time simulations since real time simulations reduce the message transmission delay overheads which are occurred in virtual time simulations.

9.3 Discussion

In our experiments, we assume that every machine, in which DEVS *Distributor* models are located, keep the same large-sized PES data which depict generic network behavior SES. However, it takes time to copy the large sized PES data from a host which monitors and captures network activities to computers which include DEVS *Distributor* models. Then, each *Distributor* deploys partitioned and customized data to *Analyzer* models which are assigned to different machines. Data flows are explained in figure 60. The dotted-line arrows present data flows of copying original PES data from the *Distributor 1* machine to the *Distributor 2* and *Distributor 3* machines.

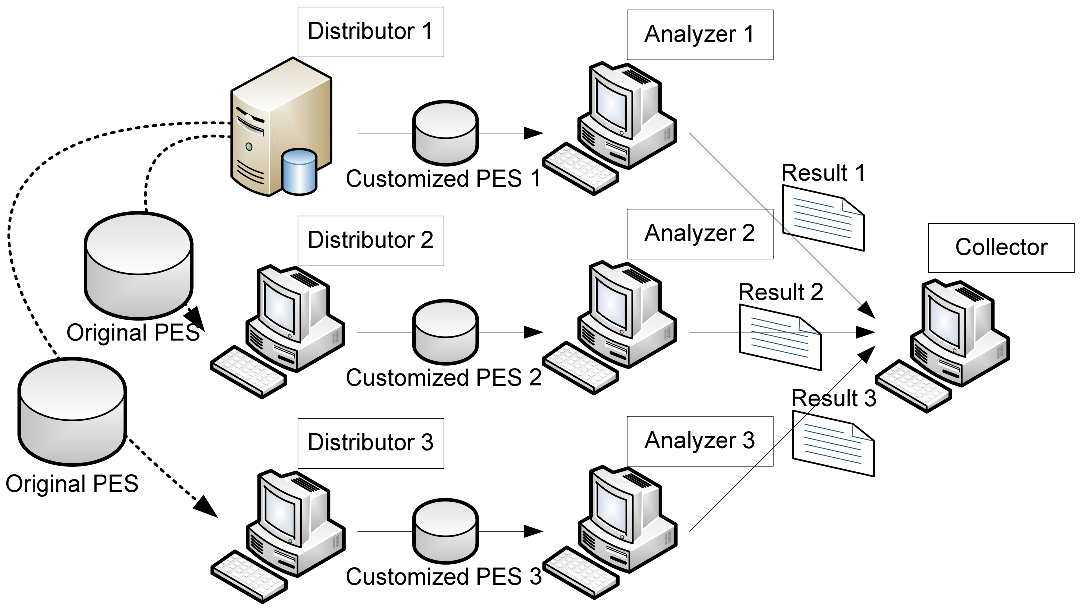


Figure 60. Semi multi-distributing and multi-analyzing machines simulation

To remove copying data time, we do experiments another approach. Three *Distributor* models are assigned into one single machine and three *Analyzer* models are located in three different computers. The data flows are shown in figure 61. This approach removes time overheads of copying original PES data time from a network activity capturing machine to the other distributing computer. However, many workloads are assigned into one distributing machine. Even though three Distributor models run on own simulator threads, operating systems do not support multi-thread processing. Therefore, only one *Distributor* model sends one message out to an analyzing computer at one time.

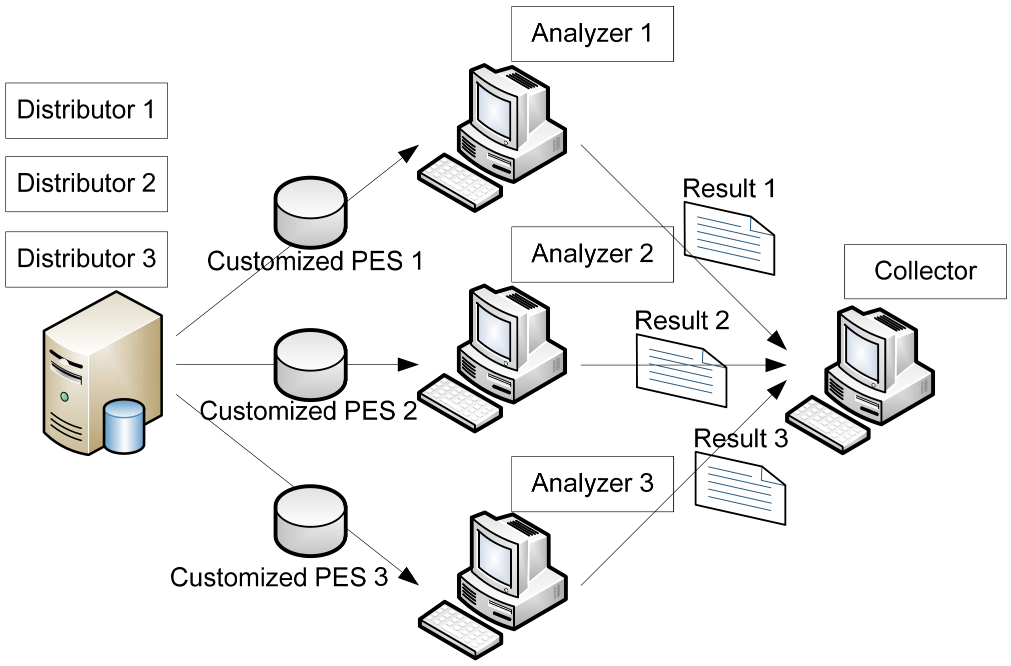


Figure 61. Single-distributing and multi-analyzing machines simulation

Execution times are shown in table 18. Two days data is evaluated in a 1*3 environment and 3*3, and, then, execution times are compared with the experimental results of the 3*3 environment presented in chapter 8.1. Although 3*3 simulation requires copying data time, total execution times (sum of copying time, distributing time, and analyzing time) of this approach is faster than total execution times (sum of distributing time and analyzing time) of 1*3 approach in both protocol analysis and throughput analysis.

Analysis	Protocol Analysis		Throughput Analysis	
Execution Times	1*3	3*3	1*3	3*3
Copying time	0	10min 49sec	0	10min 49sec
Distributing time	37min 00sec	7min 37sec	40min 57sec	10min 4sec
Analyzing time	36sec	36sec	28sec	37sec
Collecting time	1sec	1sec	1sec	1sec
Total	37min 37sec	19min 3sec	41min 26sec	21min 31sec

Table 18. Execution times of 1*3 and 3*3 environmental simulation

The experimental results illustrated in table 18 provide us feasible approach to fast overall analysis time. We may speed up by combining these two approaches (1*n and n*n). Deploying multiple computers inside sub-networks and assigning capturing network activity roles to the computers facilitate to remove copying PES data times (because every computer keeps their own captured data) and reduce distributing customized PES data times (since every *Distributor* model is distributed into individual machines). Figure 62 shows this approach. Because every distributing machine captures different network activities, a *Collector* model must have intelligent gathering methods to integrate different analyzed results together.

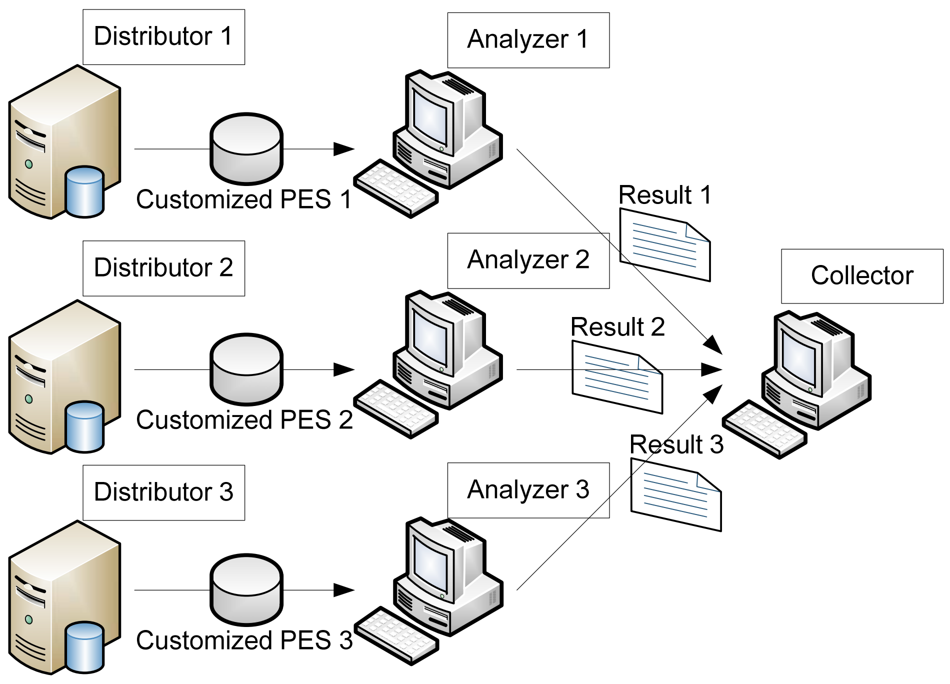


Figure 62. Multi-distributing and multi-analyzing machines simulation

CHAPTER 10. CONCLUSIONS AND FUTURE WORKS

10.1 Conclusions

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 network traffic analysis tools that could produce results quickly and accurately. There are several network traffic analysis tools such as tcpdump, Ethereal, and other applications. But, these tools have drawbacks: limited data size and excessive resource consumptions. These problems cause a slow analyzing time and require big budgets for maintaining. In addition to these problems, currently existing tools are limited to be performed inside networks, due to security issues. Dump files which are monitored and captured by these tools includes secure information such as user ID, passwords, and other information. These secure attributes must be protected against abnormal accesses, so observing network activities from out of networks should be prohibited. However, network behaviors need to be analyzed outside target networks in some cases.

This study proposes a Web-based distributed simulation for network traffic analyses over Service Oriented Architecture (SOA). The main objective of this study is 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 System Entity Structure (SES) theory is applied. The SES facilitates implementing a system to achieve our main goal, fast and accurate network traffic evaluation. The SES is a theory for designing structured information hierarchically and efficiently. Specifically, the SES is very useful for data engineering for a high throughput and a low response time. The SES data engineering helps SES/NZER to transform captured network traffic data into practical data expressed by Extensible Markup Language (XML). The advantages of the SES using a data engineering concept are easy and fast access to information, no necessary for multi-query, fast response time, and a user-centric schema. Also, the SES data engineering is a hierarchical tree structure, and the hierarchical tree structure is easy to read, understand, and manipulate. We design a generic network behavior in SES format. Also, automated awareness to pragmatic frames (customers' applications) makes reactions fast and results in no needs of human interference. We must notice that every customer has different requests (different applications). For example, some customers want to evaluate network protocol uses. Other 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 for creating a new SES to correspond to users' requests by pruning operations and mapping the newly generated SES with the pre-defined SES which represents a generic network packet behavior enables systems to be adaptively optimized. 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 enable large amounts of data to be examined, we still need a long evaluation time. To speed up evaluation time, we apply a Web-based distributed simulation approach over Service Oriented Architecture (SOA). Deploying workloads into multiple machines decreases burdens of individual computers, and results in hosts, which have low computational powers (CPU and memory), to participate in large scale simulations. As a result, there are no needs for super computers anymore. DEVS/SOA (DEVS/Service Oriented Architecture) facilitates deploying workloads into multi-servers, and, consequently, increasing overall system performance.

10.2 Future Works

We achieve both evaluating large amount of network traffic activity data and performing a Web-based distributed simulation over SOA. In addition, we accomplish fast execution times through real time decentralized distributed simulation. However, there are further research works: developing web services for network traffic analyses and implementing additional attack detecting functions for intrusion detection systems. The ultimate goal is to implement network behavior analyses web services. This study aims for a decentralized distributed DEVS simulation to speed up evaluation times by deploying workloads into multi-computers. But, customers are still responsible for building models for simulating their systems. Web services, which are implementations of integrating automated model constructing process with analyzing corresponding system process, provide more accommodation to users. Another future work is implementing web service systems to perform the case that customers hold data which need to be analyzed. Customers may provide data to multiple web services asynchronously. Subsequently, web services evaluate received data and give evaluated results back to customers.

REFERENCES

1. Teorey, T.J., D. Yang, and J.P. Fry, *A logical design methodology for relational databases using the extended entity-relationship model.* ACM Computing Surveys (CSUR), 1986. 18(2): p. 197-222.

2. Sarjoughian, H. and R. Flasher, *System Modeling with Mixed Object and Data Models, DEVS Symposium*, in *Spring Simulation Multiconference*. 2007: Norfolk, Virginia. p. 199-206.

3. Gruber, T.R., *A Translation Approach to Portable Ontology-Specifications.* Knowledge Acquisition Journal, 1993. 5(2): p. 199-220.

4. IEEE Technical Committee on Data Engineering (TCDE). 2007, http://tab.computer.org/tcde/

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

6. Foundation For Intelligent Physical Agents (FIFA). *An Ontology Service Specification*. 2007, http://fipa.org/specs/fipa00086/XC00086D.html#_%20Toc505571321

7. W3C. *eXtensible Markup Language XML*. 2007, http://www.w3.org/XML/

8. SESBuilder. *An Integrated Tool to utilize System Entity Structure*. 2007, http://www.sesbuilder.com/

9. Zeigler, B.P., *Multi-Facetted Modeling and Discrete Event Simulation*. 1984, New York: Academic Press.

10. Zeigler, B.P. and G. Zhang, *The System Entity Structure: Knowledge Representation for Simulation Modeling and Design.* Artificial Intelligence, Simulation and Modeling, L. E. Widman, K. A. Loparo, and N. R. Nielsen, Eds. Wiley, 1989: p. 47-73.

11. Harold, E.R. and W.S. Means, *XML in a Nutshell - A Desktop Quick Reference*. 2001: O'Reilly.

12. Park, S., *Cost-Based Partitioning For Distributed Simulation of Hierarchical Modular DEVS Models*, in *The Department of Electrical and Computer Engineering*. 2003, the University of Arizona: Tucson.

13. Chandy, K.M. and J. Misra, *Dsitributed Simulation: A Case Study in Design and Verification of Distributed Programs.* IEEE Transactions on Software Engineering, 1979. 5: p. 440-452.

14. Bryant, R.E., *A Switch-Level Model and Simulator for MOS Digital Systems.* IEEE Transactions on Computers, 1984. 33: p. 160-177.

15. Jefferson, D.A. and H. Sowizral, *Fast Concurrent Simulation Using the Time Warp Mechanism*. 1985, The Society for Computer Simulation (SCS): La Jolla, California.

16. Jefferson , D.A., *Virtual Time.* ACM Transactions on Programming Languages and Systems, 1985. 7: p. 404-425.

17. Zeigler, B.P., T.G. Kim, and H. Praehofer, *Theory of Modeling and Simulation 2nd ed.* 2000, New York: Academic Press.

18. Cho, Y.K., et al., *Design Consideration for Distributed Real-time DEVS*, in *AI and Simulation Conference*. 2000: Tucson, AZ.

19. Champion, M., et al. *Web Services Architecture*. 2002, http://www.w3.org/TR/2002/WD-ws-arch-20021114/

20. W3C. *Simple Object Access Protocol (SOAP)*. 2008, http://www.w3.org/TR/soap/

21. W3C. *Web Service Architecture*. 2008, http://www.w3.org/TR/ws-arch/

22. W3C. *Web Services Description Language (WSDL)*. 2008, http://www.w3.org/TR/wsdl20-primer/

23. UDDI. *Universal Description, Discovery and Integration (UDDI)*. 2008, http://uddi.xml.org/

24. SUN. *Service-Oriented Architecture (SOA)*. 2008, http://www.sun.com/products/soa/index.jsp/

25. REST. *Representational State Transfer (REST)*. 2008, http://rest.blueoxen.net/cgi-bin/wiki.pl

26. W3C. *Web Ontology Language OWL*. 2007, http://www.w3.org/TR/owl-features/

27. W3C. *Resource Description Framework RDF*. 2007, http://www.w3.org/RDF/

28. W3C. *Resource Description Framework Schema*. 2007, http://www.w3.org/TR/rdf-schema/

29. OKBC. *Open Knowledge Base Connectivity*. 2007, http://www.ai.sri.com/~okbc/

30. OMG Group. *The Unified Modeling Language (UML), Version 2*. 2007, http://www.uml.org/

31. OMG Group. *Object Modeling Group (OMG)*. 2007, http://www.omg.org/

32. Wikipedia. *Unified Modeling Language*. 2007, http://en.wikipedia.org/wiki/Unified_Modeling_Language

33. Stanford University. *Protege Ontology Editor and Knowledge-base Framework*. 2007, http://protege.stanford.edu/

34. Stanford University. *Protege Frame Editor*. 2007, http://protege.stanford.edu/overview/protege-frames.html/

35. Stanford University. *Protege OWL Editor*. 2007, http://protege.stanford.edu/overview/protege-owl.html/

36. MusicBrainz. *Music Metadata System*. 2007, http://musicbrainz.org/

37. Swartz, A., *MusicBrainz: A Semantic Web Service.* IEEE Intelligent Systems, Intelligent Web Services, January/February 2002. 17(1): p. 76-77.

38. The United Nations Standard Products and Services Code (UNSPSC). 2007, http://www.unspsc.org/

39. Cost, S., et al., *ITTALKS: A Case Study in the Semantic Web and DAML+OIL.* IEEE Intelligent Systems, Intelligent Web Services, January/February 2002. 17(1): p. 40-47.

40. DARPA. *DARPA agent markup language website*. 2007, http://wwww.daml.org/

41. Hendler, J. and D. McGuinness, *The DARPA agent markup language.* IEEE Intelligent Systems, Intelligent Web Services, November/December 2000. 15(6): p. 72-73.

42. The UCI KDD Archive. *KDD 1999 Cup dataset.* 1999, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

43. Arizona Center for Integrative Modeling and Simulation (ACIMS). 2007, http://www.acims.arizona.edu

44. Ethereal. *network protocol analyzer*. 2007, http://www.ethereal.com/

45. Puketza, N., et al., *A Software Platform for Testing Intrusion Detection Systems.* IEEE Software, September/October 1997. 14(5): p. 43-51.

46. Puketza, N., et al., *A Methodology for Testing Intrusion Detection System.* IEEE Transactions on Software Engineering, 1996. 22(10): p. 719-729.

47. Bishop, M., S. Cheung, and e. al., *The Threat from the Net.* IEEE Spectrum, 1997. 38(8): p. 56-63.

48. Haines, J.W., et al., *1999 DARPA Intrusion Detection Evaluation: Design and Procedure*. February 2001 Lincoln Laboratory, Massachusetts Institute of Technology.

49. Mittal, S., B.P. Zeigler, and M.H. Hwang. *W3C XML Schema for Finite Deterministic(FD) DEVS Models*. 2007, http://www.u.arizona.edu/%7Esaurabh/fddevs/FD-DEVS.html

50. Mittal, S., *DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures*, in *Department of Electrical and Computer Engineering*. 2007, the University of Arizona: Tucson, AZ.

51. W3C the Document Object Model Working Group. *Document Object Model (DOM)*. 2007, http://www.w3.org/DOM/

52. JFreeChart. *Java chart library*. 2007, http://www.jfree.org/jfreechart/

53. Mittal, S., J.L. Risco-Martin, and B.P. Zeigler, *DEVS-Based Simulation Web Services for Net-centric T&E*, in *2007 Summer Computer Simulation Conference (SCSC'07)*. July 2007: San Diego, CA.

54. Mittal, S., J.L. Risco-Martín, and B.P. Zeigler, *DEVSML: Automating DEVS Execution Over SOA Towards Transparent Simulators*, in *DEVS Symposium, Spring Simulation Multiconference*. March 2007: Norfork, Virginia. p. 287-295.

55. Karatsuba, A. and Y. Ofman, *Multiplication of multidigit numbers on automata.* Soviet Physics doklady, 1963. 7(7): p. 595-596.

56. Lincoln Laboratory in Massachusetts Institute of Technology. *DARPA Intrusion Detection Evaluation*. 2008, http://www.ll.mit.edu/IST/ideval/index.html/

57. *KDD Cup 1999*. 2008, http://www.sigkdd.org/kddcup/index.php?section=1999&method=info/

58. Glinsky, E. and G. Wainer. *Definition of Real-Time simulation in the CD++ toolkit*. in *Proceedings of SCS Summer Computer Simulation Conference,*. 2002. San Diego, CA.

59. Kim, K., et al. *Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One*. in *Proceedings of the 33rd Annual Simulation Symposium*. 2000. Washington DC.

60. Lee, J.S., *Space-Based Data Management for High Performance Distributed Simulation*, in *Department of Electrical and Computer Engineering*. 2001, the University of Arizona: Tucson, AZ. p. 221.

61. Vargas, J., et al., *PDU Bundling and Replication for Reduction of Distributed Simulation Communication Traffic.* The Journal of Defense Modeling and Simulation, 2004. 1(3): p. 171-183.

62. *Apache Tomcat*. 2008, http://tomcat.apache.org/

63. *Apache Axis2 Web service engine*. 2008, http://ws.apache.org/axis2/

64. W3C. *Simple Object Access Protocol (SOAP) 1.1 Note*. 2008, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

65. Zhang, M., *Toward a Flexible and Reconfigurable Distributed Simulation: A New Approach to Distributed DEVS*, in *Department of Electrical and Computer Engineering*. 2007, the University of Arizona: Tucson. p. 137.