Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Computer Networks 43 (2003) 573–600

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

NSDF: a computer network system description framework and its application to network security

Juan M. Estévez-Tapiador [*], Pedro García-Teodoro, Jesús E. Díaz-Verdejo

Research Group on Signals, Telematics and Communications, Department of Electronics and Computer Technology, University of Granada, E.T.S. Ingeniería Informática, C/Daniel Saucedo Aranda S/N, 18071 Granada, Spain

Received 25 June 2002; received in revised form 25 April 2003; accepted 6 May 2003

Responsible Editor: S.F. Wu

Abstract

In this work a general framework, termed NSDF, for describing network systems is proposed. Basic elements of this scheme are *entities* and the *relationships* established between them. Both entities and relationships are the basis underlying the concept of *system state*. The dynamics of a network system can be conceived of as a trajectory in the state space. The term *action* is used to describe every event which can produce a transition from one state to another.

These concepts (entity, relationship, state, and action) are enough to construct a *model* of the system. Evolution and dynamism are easily captured, and it is possible to monitor the behaviour of the system. With the aim of illustrating the use of the proposed framework, a network state description language derived from NSDF, termed RENDL, is also specified.

An immediate application of this framework concerns the network security field. It is shown that concepts like security policing of the site, insecure states, intrusive activities and intrusion response mechanisms can be modelled well. Thus, some imprecise terms used in the security context can be expressed in a uniform, precise way within this framework. Formalizing the above concepts allows us to introduce a generic model to classify currently presented taxonomies related to intrusive activities in network systems. This provides a general context for a better understanding of security flaws and how to develop effective defenses.
© 2003 Elsevier B.V. All rights reserved.

Keywords: Computer network description; Computer network security; Intrusion detection; Intrusion response; Intrusion taxonomies

1. Introduction

In recent years, computer networks have become a common work environment for users and companies. The basic idea behind network systems, and the reason why they are implemented in a work environment, is the availability of services

[*] Corresponding author. Tel.: +34-958-242305; fax: +34-958-240831.
E-mail addresses: tapiador@ugr.es (J.M. Estévez-Tapiador), pgteodor@ugr.es (P. García-Teodoro), jedv@ugr.es (J.E. Díaz-Verdejo).

and resources for users. Such an availability should not be anarchical or random; on the contrary, it must be properly operated if we are to derive its potential benefits and also be aware of the risks involved. The correct operation of a given network system depends on two main aspects: (a) good planning and design from the outset and (b) constant event monitoring and management tasks to prevent and/or correct abnormal situations. Both questions, especially the second one, could be more easily dealt with from a purely technical and theoretical point of view if a formal model to describe the system existed. It is well known that event monitoring is a fundamental component of many computing activities, one that can be used for several purposes: retrieving system status information, ensuring that tools make efficient use of limited computing resources, load balancing and smooth degradation in the presence of failures, and so on [22].

Technological advances have enabled a huge growth in the functionality provided by these kinds of systems, whilst connectivity ratios grow and new services appear continually. Thus, it is common to find medium-size networks composed of numerous workstations, network elements like routers, printers, databases, services such as e-mail and file transfer, users making use of the applications, and so on. At first glance, this complex and rather heterogeneous framework may seem somewhat unattractive for developing a general, and at the same time exact, "description language" that allows us to extract and obtain an accurate outline of the most important features of a computer network system. However, it is exactly in this context where this objective is most necessary, because a more comprehensive vision and control of the system and the events related to it is required.

With this aim in mind, a general formalism is proposed in this paper. Termed NSDF, it is based on two atomic and modular concepts: *entity* and *relationship*, which permit us to define not only the various elements (hardware and software) that comprise a given network system but also the several available links or connections between them. From the enumeration of the entities in a system and their relationships, the concept of *state* is introduced. This idea is fundamental because it provides the perfect reference space in which to study the evolution of the system due to *actions*. Section 2 introduces the theoretical NSDF foundations and its use is illustrated by means of an example with which a typical network system is described in detail. A language, termed RENDL has been derived from NSDF. It is introduced in Appendix A, providing its BNF, while Appendix B shows an example of its use for describing a simple network system.

Section 3 applies NSDF to the field of computer and network security, which is of great interest nowadays. We show that it is possible to construct a theoretical model of the system and thus formalise definitions concerning the subject of security and provide a better understanding of the related phenomena. Furthermore, NSDF constitutes a general framework that permits us to unify the various taxonomies in the area of intrusion detection that have been proposed in the literature in the last few years.

Finally, Section 4 summarises the paper by presenting our main conclusions, the benefits of the work developed and future research objectives.

2. NSDF: a general framework for describing computer network systems

Computer network systems can be viewed as complex assemblies of hardware components (network interfaces, processors, wires, storage devices, etc.) and software components (operating systems, data files, applications, routing tables, configuration files, etc.). Current systems have reached such a level of complexity that it is practically impossible to obtain a characterization of them by describing all their components and properties. In addition to this, most of the interesting behaviour patterns and functions of a computer network are not only derived from its components, but also from the relationships established between them. For example, there is some kind of relationship between a network interface plugged into a computer and the protocol stack managed by the operating system, which processes incoming and outcoming information flow from the node. When an application (for example, an HTTP client) establishes a TCP connection to a remote application (an HTTP server), a relationship of connectivity between the two entities is defined. The presence or absence of these relationships, in conjunction with the elements of the system, is fundamental for describing the current *state* of the system.

Having a theoretical model of the system, namely, a formal structure that is an abstraction of the system components, means several benefits are obtained. One such benefit concerns the availability of event monitoring in the system through the evolution of its state. The second advantage is related to the better understanding of system behaviour and, especially, to the existing dependences between objects in the system. Similarly, having such a model is a fundamental requirement for the implementation of complete automatic intrusion response mechanisms. These mechanisms have to take decisions and elaborate a reasoning about which components are compromised, which elements have to be dynamically reconfigured, and so on. Performing these and other tasks requires some kind of model of the system to be created.

Any description of a computer network is highly dependent on the purposes and objectives of the model constructed. Most of the work done in this field is oriented toward network simulation, emulation and design [12,18]. Nevertheless, the approach taken in our research is based on modelling an *existing* computer network system for monitoring purposes. In this context, some characteristics are fundamental, such as capturing interactions between components and extracting dynamic information concerning networking processes. The following definitions establish the formal framework proposed in this paper.

2.1. Definitions

Definition 1 (*Network system description*). A network system description $\mathbf{S}$ is a pair $\mathbf{S} = \langle \mathbf{E}, \mathbf{R} \rangle$, where $\mathbf{E} = \{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N\}$ is the set of entities in the system, and $\mathbf{R} = \{\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_M\}$ is the set of relationships between them. The following two definitions explain the concept of entity and relationship in greater depth.

Definition 2 (*Entity*). A system can be considered as an assembly of components. An entity represents every atomic object inside the system which is of interest from the point of view of functionality and management. The entity can be described completely by the following five attributes:

$e = \langle \mathrm{Id}, N_\mathrm{L}, S_\mathrm{A}, S_\mathrm{O}, A_\mathrm{S} \rangle$

where

- Id is a *unique identifier* for the object. It allows the differentiation and unique identification of entities within the system. It could be a number or, more generally, a string of characters.
- N_L is the *network level* of the entity (see later definition for a more in-depth explanation).
- S_A is the *administrative state* of the entity, and valid values are "permitted" or "not-permitted." An entity can be temporarily disabled for administrative reasons (for maintenance purposes, for example), or its use may be currently permitted.
- S_O is the *operative state* of the entity. Valid values are "operative" or "not-operative." Regardless of the use allowed to an entity, it may be in a state of malfunction, for various reasons. If an entity is not operative, no satisfactory result can be obtained from it.
- A_S is a set of *specific attributes* of the entity, different for each class of objects and derived from the network level of the entity.

An example of an entity within a network system is an *application*, like an SMTP or an FTP server, or an HTTP client. Each of these applications is an object that performs a task. A network interface card is another example of an entity that is frequently present in hosts. More abstract entities are those related with layers of the network and protocols. For example, a TCP port can be viewed as a single unit, with its own state.

It is well known that the best way to understand a network system is to consider it as a stack of layers. Classic examples are the OSI model, which defines seven layers, or the TCP/IP model, which is well described with layers: physical, MAC, IP, transport (UDP or TCP) and the application layer (HTTP, FTP, Telnet, SMTP, etc.). For example, SNMP (Simple Network Management Protocol) is a widely used protocol for management purposes in TCP/IP networks, and it defines several managed objects by network layers. Each entity within the system is related to the network level in which it operates. For example, a network card is clearly an object at the physical layer, a port is located at the transport layer, and an FTP client is obviously in the application layer.

Thus, each entity inside the system can be associated with a layer of the network. Similarly, each network level has specific attributes that enable it to be described. For example, if we consider an entity at the physical level, then we are interested in parameters like its speed, type of interface, etc. In order to illustrate how to describe network-related information about an entity, we use a typical TCP/IP network model. In this framework, an entity is found in one of the following levels:

(i) *Physical level*: At this level, every object in the network physical layer is compounded, and immediately related to data transmission. Every entity in this level can be described by the following attributes (that is, the set A_S of specific attributes for the entity):
   - *Type*: following the description of the basic elements of a network, three types of physical entities are considered: link, interface and node.
   - *Subtype*: intended to provide special features of the three previous entities, this attribute takes the following values: point-to-point or multicast for links; serial, RJ-45, etc. for an interface; and host, printer, router, etc. for a node-type entity.
   - *Speed*: this parameter captures the speed of the entity in bps, instructions/seg, pages/min, etc.

(ii) *MAC level*: This level contains all the software components which perform the functions for access control to the medium. They are usually related with low-level transmission technology. For example, in local area networks the following specific attributes A_S can be defined:
   - *Type*: class of access technology: 802.3, 802.5, etc.
   - *Address*: hardware address of the network interface card.
   - *MTU*: maximum transfer unit of the entity.
   - *Mode*: promiscuous mode or not.

(iii) *IP level*: Entities at the IP layer comprise aspects related with the IP configuration. In this sense a single kind of entity is considered: IP addresses. Three types of IP addresses exist: unicast IP address, broadcast address, and netmask address. Hence, we propose the following three attributes for IP entities:
   - *IP*: unicast IP address of the entity.
   - *Mask*: network mask of the entity.
   - *Broadcast*: broadcast address.

(iv) *TCP/UDP level*: Ports are basic entities at the transport layer. The following parameters can be used to describe the specific attributes A_S of a port entity:
   - *Type*: indicates if it is a TCP or a UDP port.
   - *State*: port state is active (if it starts the activity, that is, if it is used by a client application), or passive (if it is used by a server application).
   - *Number*: port number in which the entity is working.

   In the case of TCP entities, extra attributes could be added, such as the actual state of the TCP state machine (ESTABLISHED, LAST_ACK, CLOSE_WAIT, etc.).

(v) *Application level*: entities at the application level can be viewed as intermediary objects between resources and other entities. For example, an FTP server is an intermediary between the FTP client and the files accessed. The client applies its requests to the application, and this entity replies with the requested file or the appropriate error message. Likewise, an HTTP server can be described in a simplistic way as an intermediary between the client browser and the requested documents. Thus, we can associate the following attributes A_S with an application:
   - *Type*: resource access mode (http, telnet, nfs, etc.).

- *Name*: name of the resource (file, directory, URL, etc.).

The concept of *network level* provides a good deal of flexibility. Defined levels, like physical, MAC, IP, TCP/UDP or application are examples that cover most of the entities, but it is possible to define new levels for other entities. For example, in the same context as above, it is possible to introduce a new level, IGMP, and to describe its relevant, specific attributes. Similarly, a general and wider level like "application" could be subdivided into several levels, in order to distinguish between different kinds of applications. This modularity is a powerful tool for describing entities.

Definition 3 (*Relationship*). As stated at the beginning of this section, a network system is a complex, dynamic group of entities which are constantly interacting. We use the term *relationship* to describe the use that one entity makes of the functions offered by another.

A relationship between two entities can be described by the following seven attributes:

$$r = \langle \mathrm{Id}, E_S, E_T, S_A, S_O, T, A_S \rangle$$

where

- Id is a *unique identifier* for the relationship. As with entities, it is necessary that relationships should be distinguished and referred to without ambiguity.
- E_S and E_T are, respectively, the *source* and the *target entities* of the relationship.
- S_A is the *administrative state* of the relationship. This field has the same meaning as the administrative state of an entity, but with some subtle differences. As before, accepted values are "permitted" and "not-permitted." In this case, an administrative state set to not-permitted means that this relationship is not allowed. For example, a Telnet connection between two hosts could be forbidden by the system administrator.
- S_O is the *operative state* of the relationship. Possible values are "operative" and "not-operative." A relationship is operative if it is currently working.
- T is the type of the relationship (see later explanation).
- A_S is the set of *specific attributes* of the relationship.

At first glance, it might seem difficult to comprehend the complex behaviour of a typical network system constituted of hundreds of entities, distributed within dozens of hosts, with several network segments, routers, etc. But due to the structured and well defined levels of operation of the network, every relationship between two entities in a network system can be classified into two types:

(i) *Vertical relationships*: These are *support* relationships, that is, those in which one entity needs another in an adjacent network layer in order to perform its function. For example: an MAC entity is supported by a network physical interface, an application entity is supported by a TCP or a UDP port, and so on. This type of relationship is related to the nature of the user/provider of a service to an entity.

(ii) *Horizontal relationships*: These are *accessibility* relationships between peer-remote entities in the same network level. For example, a network can be accessed by a host, one IP address can be accessed by another, a TCP port can be read or written to from another TCP port, and so on.

Each relationship (regardless of whether it is vertical or horizontal) has three specific attributes: *read*, *write* and *execute*, which can be enabled or disabled. Fig. 1 illustrates both types of relationships between several local and remote entities within a system. With no loss of generality, in a complex network system it is easy to identify partially isolated groups of entities and relationships between them which, *together*, perform a task. We use the term *subsystem* to denote such structures. For example, let us analyze an application such as an HTTP server running on a host. The server application, which is an entity at the application layer, uses a port (typically port 80) at the TCP level.

This port is an entity at the transport layer, and a support relationship exists between this port and

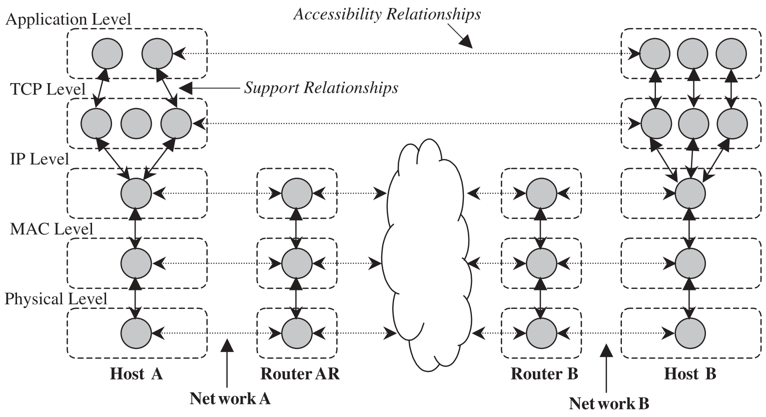

Fig. 1. Example of relationships between entities at different network levels in a basic TCP/IP internetworking environment with two hosts and two routers.

the application server. Additionally, this port is supported over an IP entity which comprises the IP configuration (IP address, mask, etc.). Therefore, we can identify another support relationship between the two. Moreover, the IP entity is operating over an MAC object (say, a typical Ethernet configuration), and this, in turn, is operating over a physical netword card. Fig. 2 illustrates schematically the presence of four subsystems on a host with two network interface cards.

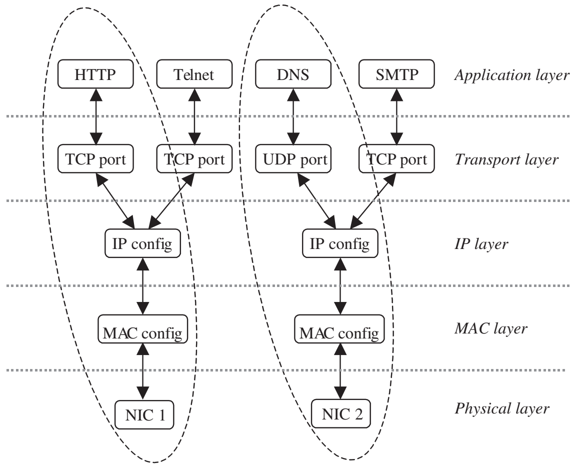

Fig. 2. Example of four subsystems on a host with two network interface cards (NIC). Two of these are indicated with dashed lines. Each subsystem is composed of one entity at each network layer, and a set of support relationships between them. Several different subsystems can share entities at some levels.

Definition 4 (*System state and state space Σ*). For every object in the system (entity or relationship), it is possible to define its state directly from its attributes. Thus, we can define the state of a network system at a given moment as the set of entities present in it, the set of relationships between them, and the states of each object in each of the sets. Evidently, the entire system can be described immediately by its state. Note that a change in the state of an entity, or the appearance or disappearance of a relationship produces a change in the state of the entity or the relationship, respectively, and hence, in the entire state of the system.

From this concept and the above definitions, we can formulate the concept of *state space* Σ for a network system such that the evolution of the system can be understood as a movement, or trajectory, across this space. Σ is thus defined as

$$\Sigma = \{s_1, s_2, \ldots, s_P\}$$

where each s_i is a different state. It is important to note that a state is not only configured by the presence or absence of the relationships between entities in the local system, but also by the relationship between them and the entities in remote systems.

It is possible to establish that two states s_i and s_j are equal if, and only if, they have the same entities with the same attributes, and the same relationships with the same attributes (regardless of whether they are between local or remote entities).

Definition 5 (*Action*). An action is any event that can modify the state of the system and so cause a transition in the state space. This definition is similar to that provided in the IEEE Standard Dictionary for *event*: "*an action directed at a target which is intended to result in a change of state* (*status*) *of the target*" [11]. Every change in the state of a system is due to an action produced on it (see Fig. 3), and can be represented in a hierarchical way by an ordered set of events for every network layer, down from the upper level event to the physical level. For example, suppose that an FTP client establishes a connection with a server. This action can be described at the TCP level as a new horizontal relationship between the client and the server, both of which are entities with their own respective attributes (port number, port state, etc.). But these entities have a previous vertical relationship with their respective IP entities, which support them (IP address, netmask, etc.).

Thus, we can view this action at TCP level as the *TCP network traffic* which originates the creation of the new relationship. More specifically, this traffic is supported by *IP traffic*, that is, by a set of packets corresponding to the action at the IP level, and so on. For every network level it is possible to give a set of actions capable of establishing new relationships between entities at that level, or that can modify existing ones. Fig. 4 illustrates this concept. It is important to note that a single action at the application level (for example, an HTTP request) can be supported over more than one action at inferior levels (more than one IP datagram).

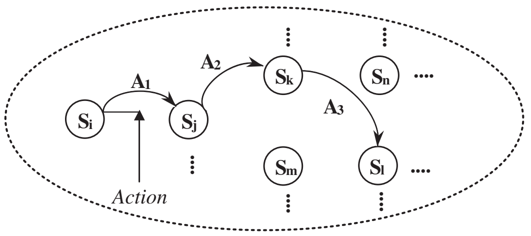

Fig. 3. Illustration of actions as events that produce transitions into the state space. Every action produces a transition from one state to another.

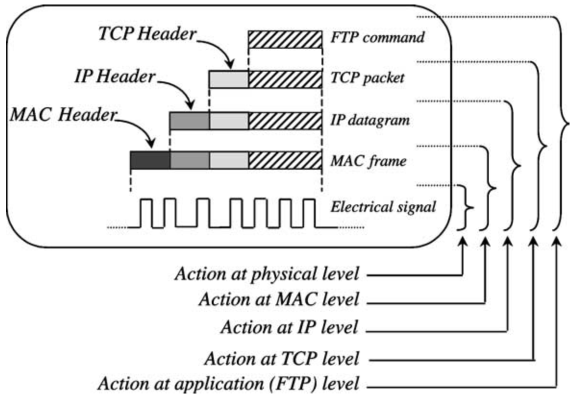

Fig. 4. Conformation of an action at the application level corresponding to an FTP command. Each action at a level is generated by an entity at that level. Due to the support relationships, every single action can be described as one or more actions at inferior levels in the hierarchy (for example, a TCP packet can be transmitted in two IP datagrams).

The five definitions given above establish a formal framework and a complete set of tools for describing network systems. A major advantage of this scheme is its property of *granularity*. As stated above, systems can be excessively complex, and a fine-grain description (that is, a description of every piece in the system and its relationships) could be unmanageable in most of the senses. Sometimes it is recommended that a block of entities and internal relationships should be considered as a single entity, even if a loss of power of description, and hence information, is lost. By this means it is possible to focus the model on the objects of interest at the appropriate level of description. This property provides enough flexibility to adjust the scale of description to the requirements of the problem.

Complexity could increase with the number of entities and corresponding relationships in a system description, which in turn will render the applicability of the proposed model. The key concepts to solve this drawback are *abstraction* and *encapsulation*, as well as *composition*. Simply put, an entity constitutes an atomic object, i.e., an item without internal structure. An entity is only described through an unique identifier and a list of attributes. We can thus define recursively a system as a list of entities as well as a list of systems.

Therefore, abstraction and encapsulation are provided by the notion of atomic object, which is directly related to the granularity of description. For example, we can describe a system assuming that an entity is a whole host, with its own list of specific attributes containing the desired information. From this, it is possible to describe more complex structures like clusters of hosts by establishing relationships between those entities.

However, in other applications it could be desirable to maintain a fine-grain description, in such a way that, for example, each host will be a system with structure (entities, or even more complex subsystems inside, and relationships among them). Moreover, note that NSDF allows us to maintain simultaneously heterogeneous descriptions from granularity level point of view.

The second remarkable property of NSDF is its *scalability*. The description of the system can be easily extended when new entities are added.

Having presented the basic terminology, the next section illustrates the use of this framework, by describing a simple computer network system. In addition to this, Appendix A shows an example of a simple language, termed RENDL, which can be derived from the presented framework.

2.2. An example

In this section we describe a simple computer network system using the formalism described previously. Let us suppose a simple internetworking environment as shown in Fig. 5. This is composed of two LANs interconnected by a router R. Four hosts (named A, B, C, and D) are distributed among the two networks as shown in Fig. 5.

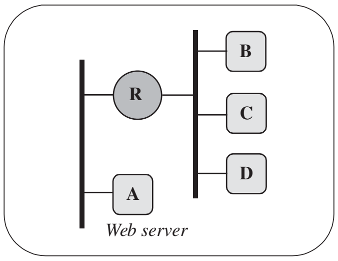

Fig. 5. A simple TCP/IP internetworking system composed of two LANs, one router and four hosts, one of which offers a Web service.

Let us suppose that host A is running a Web server. This subsystem is composed of the entities and relationships summarized in Table 1.

In the same way, Table 2 contains the basic entities and relationships corresponding to router R. This is an element with two network interface cards, and can be described by two interconnected subsystems (one for each network interface).

Besides these components, there are two entities at the physical level inside the system: the network wires. It is possible to identify two wires. The first of these is the Ethernet segment to which host A and one network interface card of router R are connected. Table 3 illustrates the two entities and their relationships with the other entities described previously.

Let us suppose a Web client is running in host B and accessing pages offered by the previous Web server. Table 4 illustrates the description of the Web client and the relationships in this subsystem. The relationships between this and the Web server subsystem are also composed of several relationships within the router R_2. Finally, Fig. 6 summarizes all the entities and relationships presented. For the interested readers, this same example is also described using RENDL language in Appendix B.

3. NSDF applied to computer and network security

As the use of computers has increased massively in recent decades, the protection of data and system resources against malicious intervention has become a field of ever-greater importance. Computer and network security is a subject that has aroused interest from the very beginnings of electronic computation, but with the solid implantation of Internet in the economic sector and, especially with its widespread domestic acceptance, concern about security matters has increased spectacularly. It is undeniable that Internet and communication networks are, nowadays, more than a simple daily fact. Our society has created such a dependence on telematic systems that their security has become, not just a matter of public interest, but a necessity to guarantee the future development and technological integration

Table 1
Entities and relationships in the Web server subsystem

Description	Id	N_L	S_A	S_O	A_S
Entities in the Web server subsystem at host A					
Network interface card at host A	1	Physical	Permitted	Operative	Type = interface Subtype = RJ-45 Speed = 100 Mbps
Medium access control for the interface	2	MAC	Permitted	Operative	Address = 00-50-DA-B9-63-AA Type = IEEE 802.3 (Ethernet) MTU = 1500 bytes Mode = non-promiscuous
IP configuration for the interface	3	IP	Permitted	Operative	IP = 150.214.60.223 Mask = 255.255.255.0 Broadcast = 150.214.60.255
Port at transport level	4	TCP/UDP	Permitted	Operative	Type = TCP State = passive Number = 80
Application Web server	5	Application	Permitted	Operative	Type = HTTP Name = /wwwroot

Description	Id	E_S	E_T	S_A	S_O	T
Relationships between entities in the Web server subsystem at host A						
Relationship between entities 1 and 2	1	1	2	Permitted	Operative	Vertical
Relationship between entities 2 and 3	2	2	3	Permitted	Operative	Vertical
Relationship between entities 3 and 4	3	3	4	Permitted	Operative	Vertical
Relationship between entities 4 and 5	4	4	5	Permitted	Operative	Vertical

of ever more facets of society. The statistics provided in [6] reveal the impressive growth of the number of computer security incidents over the last 15 years. This growth is only comparable with the expansion of communication networks and mass access to them. Thus, security in computers and communication networks has developed from something of auxiliary value to become a fundamental part of the design space.

As was pointed out in the introduction of this article, constant event monitoring is a crucial point in current network infrastructures. In order to achieve this objective and supervise the system state it is necessary to have a system description in a way useful for its processing. Section 3.1 discusses the application of NSDF for this purpose.

Besides this potential use in network monitoring and intrusion detection, the theoretical model has other applications in this field. Sections 3.2 and 3.3 illustrate two contributions to the computer and network security field. The first of them is related to the definition of some security concepts within this framework, while the second one presents a particularization of NSDF to provide a general context for unifying taxonomies on intrusive activities.

3.1. System state monitoring

The limitations of formal methods in computer and network security have made it necessary to develop security-oriented monitors to supervise system activity in search of security violations. Most of the traditional approaches to network intrusion detection and response systems are inspired by mechanisms for the analysis of activities,

Table 2
Entities and relationships in two subsystems inside router R

Description	Id	N_L	S_A	S_O	A_S
Entities in the subsystem 1 of router R					
Network interface card 1 at router R_1	6	Physical	Permitted	Operative	Type = interface Subtype = RJ-45 Speed = 100 Mbps
Medium access control for the interface	7	MAC	Permitted	Operative	Address = 00-50-DD-A7-45-30 Type = IEEE 802.3 (Ethernet) MTU = 1500 bytes Mode = non-promiscuous
IP configuration for the interface	8	IP	Permitted	Operative	IP = 150.214.60.68 Mask = 255.255.255.0 Broadcast = 150.214.60.255
Entities in the subsystem 2 of router R					
Network interface card 2 at router R_1	9	Physical	Permitted	Operative	Type = interface Subtype = RJ-45 Speed = 100 Mbps
Medium access control for the interface	10	MAC	Permitted	Operative	Address = 00-52-AD-B5-3F-F0 Type = IEEE 802.3 (Ethernet) MTU = 1500 bytes Mode = non-promiscuous
IP configuration for the interface	11	IP	Permitted	Operative	IP = 192.168.1.1 Mask = 255.255.255.0 Broadcast = 192.168.1.255

Description	Id	E_S	E_T	S_A	S_O	T
Relationships between entities in the two subsystems inside router R						
Relationship between entities 6 and 7	5	6	7	Permitted	Operative	Vertical
Relationship between entities 7 and 8	6	7	8	Permitted	Operative	Vertical
Relationship between entities 9 and 10	7	9	10	Permitted	Operative	Vertical
Relationship between entities 10 and 11	8	10	11	Permitted	Operative	Vertical
Relationship between entities 8 and 11	9	8	11	Permitted	Operative	Horizontal

such as incoming traffic [16]. Although this approach is not erroneous, it is far from complete, due to the differences between reality and perception [20]. For example, analysis of network traffic in search of intrusion signatures does not provide enough information in order to determine the *real* response of the target system when it has captured and processed the packets. Even if the analyzer maintains some kind of memory of the activities, implementation of the protocol stack and applications may be different at different nodes, and responses to specific packets might not be the same. Furthermore, the presence of encrypted traffic is a serious problem if security mechanisms are only based on analysing the information that flows through the network.

We strongly believe that, in order to perform a complete supervision of network activity, it is necessary to monitor not only incoming traffic but also the state of the system itself. In other words, to analyze the stimulus as well as the response generated by the system. The entities that make up a system, and the relationships between them, are mechanisms that can indicate the state of the system, how it has reached such a state, the threats facing the system, etc. Justification for this paradigm comes from a simple but increasingly worrying fact: the rate of growth of current networking

Table 3
Entities that represents the two network wires present inside the system

Description	Id	N_L	S_A	S_O	A_S	
Entities: network wires in the system						
Wire to which host A is attached	12	Physical	Permitted	Operative	Type = link Subtype = multicast Speed = 100 Mbps	
Wire to which hosts B, C, and D are attached	13	Physical	Permitted	Operative	Type = link Subtype = multicast Speed = 100 Mbps	
Description	Id	E_S	E_T	S_A	S_O	T
Relationships between wires and entities at Web server and router R						
Relationship between entities 1 and 12	10	1	12	Permitted	Operative	Horizontal
Relationship between entities 6 and 12	11	6	12	Permitted	Operative	Horizontal
Relationship between entities 9 and 13	12	9	13	Permitted	Operative	Horizontal

environments, together with the increasing complexity of their behaviour, makes the management of these infrastructures, and more precisely the supervision of evolution over time, crucial for network security in the near future.

The formalism described in the previous section, coupled with RENDL language defined in Appendix A, could provide the basis for constructing a theoretical security-oriented description model of a network system. The main purposes of such a model are

1. To describe, from a security point of view, a network system, its components, and how they interact. This symbolic network representation is a basic requirement for any *state monitoring* tool. For example, construction of complex queries should be allowed in order to obtain useful information like: total or partial network inventory, state of selected subsystems, etc.
2. To keep track of the state of the network system, that is, to maintain registers related to data of interest at different layers (physical, network, transport, applications, etc.). Likewise, to compare actual state and *desired* state. This comparison can be performed by using a security policy of the system, and also by detecting well-known patterns of activity which lead to security violations (*attacks*).

Our main future objective is to lead to practice this work. For achieving such a purpose, we have identified the following four tasks to be addressed:

1. To develop extensions to RENDL for describing *security policies*. For example, access control-like policies seems easily representable within this framework.
2. Similarly, a language extension for representing *attack scenarios* is also required. In this sense, one major advantage of the proposed approach is that RENDL captures easily the distributed nature of network systems, and events occurring in different places within the network can be easily represented.
3. Surely the most challenging problem is to reach a notion of *normal behaviour of a system*, that is, to have an idea concerning which entities and relationships are normal and which ones are suspicious.
4. Finally, to develop a *state monitoring tool*, whose objectives should include, although are not limited to:
   (i) *Enforcement*: to verify that network activities are within the security policy.
   (ii) *Anomaly detection*: to detect suspicious deviations with respect to normal behaviour.
   (iii) *Misuse detection*: to detect well-known events which can lead to a security breach (attacks).

Table 4
Entities and relationships in the Web client subsystem (host B), and relationships established during an HTTP request

Description	Id	N_L	S_A	S_O	A_S
Entities in the Web client subsystem at host B					
Network interface card at host B	14	Physical	Permitted	Operative	Type = interface Subtype = RJ-45 Speed = 100 Mbps
Medium access control for the interface	15	MAC	Permitted	Operative	Address = 01-72-AF-96-33-FF Type = IEEE 802.3 (Ethernet) MTU = 1500 bytes Mode = non-promiscuous
IP configuration for the interface	16	IP	Permitted	Operative	IP = 192.168.1.2 Mask = 255.255.255.0 Broadcast = 192.168.1.255
Port at transport level	17	TCP/UDP	Permitted	Operative	Type = TCP State = active Number = 1945
Application Web client	18	Application	Permitted	Operative	Type = HTTP Name = /wwwroot/index.html

Description	Id	E_S	E_T	S_A	S_O	T
Relationships between entities in the Web client subsystem at host B						
Relationship between entities 14 and 15	13	14	15	Permitted	Operative	Vertical
Relationship between entities 15 and 16	14	15	16	Permitted	Operative	Vertical
Relationship between entities 16 and 17	15	16	17	Permitted	Operative	Vertical
Relationship between entities 17 and 18	16	17	18	Permitted	Operative	Vertical
Relationship between entities 14 and 13	17	14	13	Permitted	Operative	Vertical
Relationships between Web server, router R, and Web client subsystems						
Relationship between entities 2 and 7	18	2	7	Permitted	Operative	Horizontal
Relationship between entities 3 and 8	19	3	8	Permitted	Operative	Horizontal
Relationship between entities 10 and 15	20	10	15	Permitted	Operative	Horizontal
Relationship between entities 11 and 16	21	11	16	Permitted	Operative	Horizontal
Relationship between entities 4 and 17	22	4	17	Permitted	Operative	Horizontal
Relationship between entities 5 and 18	23	5	18	Permitted	Operative	Horizontal

(iv) *Intrusion response*: to determine the optimal response facing a detected attack.

3.2. Formalization of security concepts on network systems

Terms like *security policy*, *intrusion*, *intrusion detection and response mechanisms*, or *secure/insecure state* are well known in the field of computer and network security. Nevertheless, the definition of these terms is usually imprecise. Even if the terms are commonly used and accepted, we lack a common reference framework which provides the necessary elements to establish them in a formal way. Based upon the model presented in Section 2, the following discussion shows how some of these network security terms can be defined within this context.

3.2.1. Secure and insecure states

It is natural to consider, from a security point of view, that states can be labelled as secure or

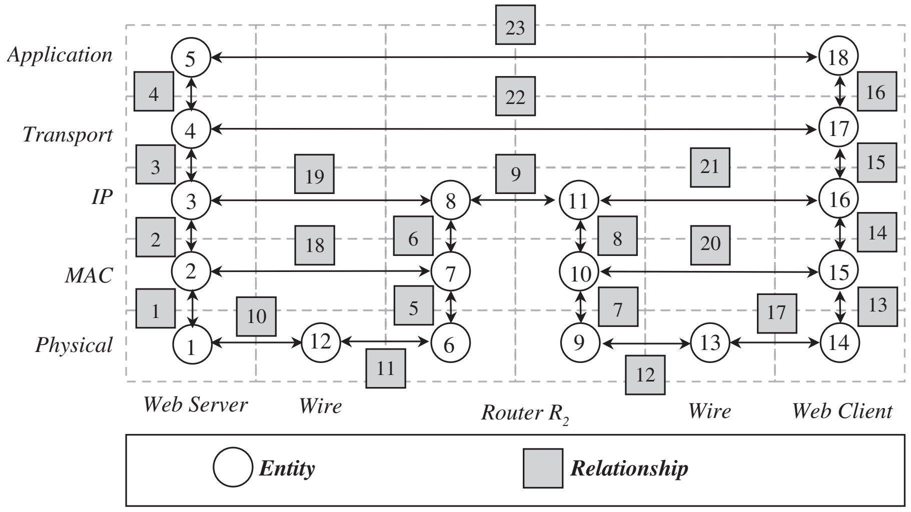

Fig. 6. Illustration of entities and relationships present in the system in Fig. 5. These represent a Web server and a client that interacts through a router.

insecure, according to the security policies of the site. For example, the presence of an established relationship between an entity in the system (say, a Telnet server) and a non-authorized remote client, which has achieved access to the system, could be considered an insecure state. The presence of certain relationships, however, is not the sole cause of the fall into an insecure state. For example, suppose that our HTTP server crashed due to a well known implementation vulnerability that is exploited by an intruder. In this case, the impossibility of establishing relationships between that entity (which could perform a valuable function in our system) and remote clients could also be labelled an insecure state.

Classic aspects of computer security are *confidentiality*, *integrity*, and *availability*. These aspects configure the model known as CIA, a brief definition of which is given in [10]: "*Confidentiality requires that information be accessible only to those authorized for it, integrity requires that information remain unaltered by accidents or malicious attempts, and availability means that the computer system remains working without degradation of access and provides resources to authorized users when they need it*."

Following this model, we can classify insecure states into three types, according to the loss of each one of the above properties. In general terms, it is possible to distinguish four regions in the state space Σ (see Fig. 7):

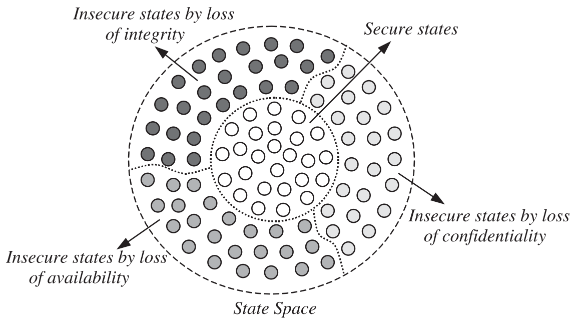
Insecure states by loss of integrity
Secure states
Insecure states by loss of confidentiality
Insecure states by loss of availability
State Space

Fig. 7. Graphical illustration of the state space of a network system. It is possible to distinguish between secure and insecure states, according to the secure policies implanted by system administrators. Insecure states are classified according to the CIA (Confidentiality, Integrity, and Availability) model. It is important to note that frontiers between groups of states are far from clear-cut, and that the representation is only conceptual.

$\Sigma = S \cup I_C \cup I_I \cup I_A$

where

- S is the set of *secure states*.
- I_C is the set of *insecure states by loss of confidentiality*.

- I_I is the set of *insecure states by loss of integrity*.
- I_A is the set of *insecure states by loss of availability*.

It is important to note that these four regions do not necessarily occupy a unique partition of the state space. Hence, it is possible for a certain insecure state to imply, simultaneously, loss of integrity and loss of confidentiality.

3.2.2. Security policy

From the above definition of secure and insecure states, it is straightforward to conceive that the security policy of a system is the set of mechanisms that, in some way, avoid transitions from secure regions of states to insecure ones. For example, when a system administrator installs a filter on a firewall that forbids incoming ICMP traffic, that mechanism has the function of avoiding the establishment of some kind of relationship which could derive to an insecure state.

There is a clear relation between the concept of security policy and the subset **S** of the state space Σ (that is, the region of secure states). However, as stated above, *frontiers between groups of states are far from clear-cut*. This implies that is very difficult, if not impossible, to classify without ambiguity a given state as secure or insecure. Hence, preventive security mechanisms cannot be enough, because of the impossibility of knowing, a priori, what exactly are the set of insecure states. This limitation can be considered one of the justifications for intrusion detection systems.

3.2.3. Intrusion, attack, or security violation

An appropriate definition of *intrusion* in the context of information technologies is provided by Amoroso [2]: "*a sequence of related actions by a malicious adversary that results in the occurrence of unauthorized security threats to a target computing or networking domain.*" Within the framework provided by NSDF, an intrusion can be defined as a set of actions:

$$I = \{A_1, A_2, \ldots, A_k\}$$

against a system which is at state S_i, producing the sequence of transitions:

$$S_i \xrightarrow{A_1} S_j \xrightarrow{A_2} \cdots \xrightarrow{A_k} S_q$$

where S_i is a secure state, and S_q is an insecure state. This sequence of transitions can be viewed as a movement from a secure region of states to an insecure one.

3.2.4. Intrusion detection mechanisms

Intrusion detection mechanisms are activities oriented to detecting intrusions, that is, the sort of trajectories on the state space as described above. In such mechanisms, there is assumed to exist knowledge about secure and/or insecure states. For example, signature-based approaches to intrusion detection analyze activities in the system in order to search for well-known malicious actions. Each detection pattern that is used embodies knowledge about one or more insecure states, usually in the form of the actions that could move the system to them.

On the other hand, anomaly-based intrusion detection mechanisms begin with a model of the *secure* behaviour of the system, and perform their function by seeking activities that constitute a deviation from this model. Such a model is obviously a form of knowledge about the set of secure states of the system.

3.2.5. Intrusion response mechanisms

The main purpose of intrusion response mechanisms is to minimize the consequences of the intrusion, thus limiting exposure to damage. According to the previous context, these mechanisms can be conceived of as actions intended to return the system to a secure state or, at least, to minimize the risks of the intrusive action.

These definitions show how a formal model of the system can help to express in a more precise way some notions of the intrusion detection field. Having this framework enables a better comprehension of fundamental ideas because of the knowledge about the system operation that it provides. In general terms, this contribution is manifest not only in the concepts defined, but in every facet in which a description of system components and their interactions is required for more precise understanding.

3.3. Application to taxonomies on computer network intrusive activities

Several taxonomies related to intrusive activities in computer networks have been proposed during the last 30 years. The need for such works is justified by the fact that classifying objects is one of the preliminary steps in learning something about them. In the words of Landwehr et al. [14]: "*A taxonomy is not simply a neutral structure for categorizing specimens. It implicitly embodies a theory of the universe from which those specimens are drawn. It defines what data are to be recorded and how like and unlike specimens are to be distinguished.*" Amoroso defines the term *intrusion taxonomy* to be a "*structured representation of intrusion types that provides insight into their perspective relationships and differences*" [2].

The central concept concerning intrusive activities is the idea of *attack*, and all existing taxonomies have tried to study it from different points of view, considering several dimensions of the phenomenon. The two main subjects of study have been the intrusion technique, that is, the set of activities carrying out the attack against a target, and the vulnerability itself, present within the system and which allows the intrusion. Moreover, and derived from the classical aspects of computer security (confidentiality, integrity and availability), some work has been done on the study of intrusions from the perspective of the result of the attack (that is, according to the aspect that has been violated). Finally, some researchers have shown the need for automatic response mechanisms when an intrusion is detected. Thus, several classifications on intrusion responses have been proposed, related to the nature of the intrusion, the vulnerability exploited, etc.

Regardless of the structure of these taxonomies (list of terms or categories, matrices/trees, etc.), all of them study the notion of attack. Nevertheless, the lack of a descriptive model of a network system implies the lack of a specific, technical framework for the study of such a concept. Different facets of this phenomenon and interesting classification criteria are always derived from a conceptual model for the attack process. But, more specifically, an attack is just a set of actions within the system, that is, commands, network traffic, access to remote services, etc. Thus, in addition to the conceptual modelling of an attack as a process, it is necessary to have a formalism that captures the essential facets of networking activities, in order to achieve an appropriate tool to construct taxonomies.

For this purpose, we propose the use of the NSDF formalism described in Section 2. The following explanation illustrates how, from a general model of the network system interactions, it is easy to derive all the facets involved in the study of an attack process.

What is known as *attack* is modelled as a process (see Fig. 8) that can be suitably described using the above terms and definitions. The elements of this process constitute a general framework that provides both a complete scheme for reorganizing current classifications, and also a functional model that allows a better understanding of network security.

The basic elements, or *dimensions* of the process are the following:

(i) *Source system* S_S, *or attacker*: There are several interesting facets related to the source of the attack. First, behind every attack there is normally an individual launching it, let us say, the originator. Some researchers have identified broad categories of attackers, according to

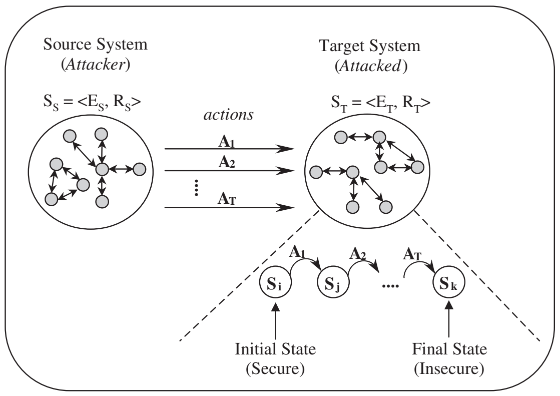

Fig. 8. General framework for classifying computer network security taxonomies in the context of NSDF, based on the idea of considering an attack as a process.

different criteria. For example, it is possible to distinguish between *internal* and *external* attackers, depending on whether the originator is a valid user of the local system or not [21]. From another point of view, it is possible to classify intruders by what they do, or, more specifically, by their motivation. Thus, there are six broad categories of attackers [10]: *hackers* (whose main interest is to achieve access to the system), *spies* (interested in the theft of information for political gain), *terrorists* (who break into computers primarily to cause fear), *corporate raiders* (whose main motivation is financial gain), *professional criminals* (who break into computers for personal financial gain), and *vandals* (whose main interest is only to cause damage). Finally, Cohen identifies some categories of intruders according to different criteria [7].

From an alternative approach, a second interesting point related to the source of the attack is the nature of the entities (in the sense of this framework) that perform the attack. Thus, it is possible to discriminate between two criteria. Firstly, according to whether the entities are *internal* or *external* to the attacked system, that is, if the attack is originated inside the system or it comes from an external system. Secondly, and according to the size and distribution of the attack, it is possible to identify *centralized* and *distributed* sources.

(ii) *Intrusion technique*: We use the term *intrusion technique* to refer to the way in which the intrusion is achieved. Intrusions are performed through *actions* between different entities inside the source and target systems. As was stated in a previous definition, an intrusion can be considered as an ordered sequence of actions, in such a way that it produces transitions of the target system from a secure state to an insecure one. It is possible to identify at least two different aspects related to the intrusion technique:

  (a) *The vulnerability exploited*: This is intimately related to the sequence of states undergone by the target system. These determine which is the security flaw that allows the movement from a secure region to an insecure one in the state space. For example, it might be easy to identify if the vulnerability is due to an implementation error of an entity (there are many well known *bugs* in applications, and administrators do not always have the most recent versions or apply the correction patches). An incorrect configuration of services could be another path used to perform an attack. This could allow an unwanted sequence of states of the system with lethal results. Several taxonomies have focused mainly on this facet. For example, Landwehr et al. [14] presented a taxonomy exemplified with many case studies in different environments. Beizer constructed a taxonomy of bugs related to the software development process [4]. Other interesting contributions in this respect can be found in [3,9,19].

  (b) *The action type*: This is concerned with the specific actions that carry out the attack (user commands, network traffic, etc.), and hence, with the appearance or disappearance of entities (or changes in their attributes) and, especially, relationships. The characteristics of the actions included in the attack allow us to classify them into several categories. For example, some actions can be labelled as *spoofing* due to the falsification of their sources. A flood of TCP packets requesting a connection to a service could be considered part of a *denial of service* attack. Of course, attacks could form a complex set of actions, and sometimes the distinction between legal and illegal actions is not clear.

  Several papers have been published on the taxonomization of these kinds of computer and network misuse techniques. Neumann and Parker [17] classified them into nine classes. Lindqvist and Jonsson proposed another taxonomy, based on an earlier one by Neumann and Parker [15]. Finally, another remarkable work in this sense was done by Kumar, focused mainly on the patterns that an action leaves in the audit trails [13].

(iii) *Intrusion result*: The intrusion result can be completely defined by the final state reached by the attacked system and its characteristics. From a point of view derived from the state space, it is obvious to classify it according to the region which includes that state. So, the result of an attack could be: (1) *corruption of information*: alteration of data stored within the system or in transit across the network [1]; (2) *disclosure of information*: access to information by anyone who is not authorized to do so [1]; and (3) *denial-of-service*: an intentional degradation or blocking of computer or network resources [7]. Some authors have discussed this and similar classifications previously (see, for example, [7,15,21]).

(iv) *Intrusion response*: Several researchers have shown the need, not only for intrusion detection mechanisms but also for intrusion response schemes [5,8]. This necessity is derived from the fast growth of automatic attack tools, like scripts or autonomous agents. These tools carry out the actions that compose the attack at a very high speed, so fast that a human response might be insufficient. From our point of view, intrusion response mechanisms can be considered as actions oriented to return the system to a state in a secure region of the state space. These actions are executed by internal entities that try to find paths into the state space towards a secure state, and their main purpose is to break illegal established relationships, to reestablish valid relationships which have been broken, etc.

The conclusions of this section are graphically illustrated in Fig. 9. The main purpose of the proposed framework (based on the concepts of NSDF) is to derive a security-oriented model of the system. This model provides both a precise description of the components of the system and a comprehension of the dynamic processes that modify the state of the system. These elements can be viewed as the essential facets or criteria to be explored in order to construct taxonomies related to the nature of the intrusive activities.

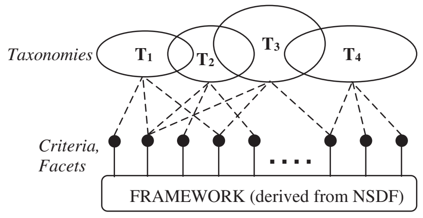

Fig. 9. Conceptual view of the framework (especially the model of the system constructed) as the basis for the existing different taxonomies of computer network intrusive activities. The model provides the elements (facets, criteria) for the different subjects of study.

4. Conclusions and future work

In this work we have presented NSDF, a generic framework for describing computer network systems and its applications to network security. NSDF provides the elements for the construction of a dynamic model for a typical network system: entities, relationships, states and actions. These four concepts are sufficient to model network systems at different levels, as well as to capture the complex interactions between their components and its behaviour. Besides its simplicity and completeness, NSDF presents the main advantage of its *scalability*. It can be used, not just to construct and formally describe a model of all the system, but especially for some subsystems or certain services that need to be monitored. Adding new entities does not imply any change to the previous model. The *granularity* is shown to be another main advantage of this scheme, allowing us the level of description detail to be adjusted to our requirements.

Section 3 presented the direct application of NSDF to network security. A theoretical model of the system allows us to define, within a formal context, diffuse notions like *intrusion*, *preventive policies*, *secure and insecure states*, *intrusion detection and response mechanisms*, etc. It provides the basis for unifying the existing taxonomies on intrusion detection within a general framework. It offers all the necessary facets related to the concept of attack, so that different taxonomies can be viewed as in-depth explorations of one or more of these aspects.

In addition to the commented points, some questions about other applications of NSDF merit discussion. In addition to the specification of RENDL (provided in Appendix A), the first natural approach is to consider the possibility of deriving from this context an appropriate language for the description of intrusions, threats, vulnerabilities, and so on. In this sense, the research will indicate the extent to which such an approach could be used, not only for a better understanding of intrusive processes, but also as a mechanism for configuring intrusion detection systems (for example, intrusion signatures, elements for describing normal and abnormal activities in the system, etc.), establishing state monitoring tools, etc. Another point is the direct application of NSDF in the field of automatic intrusion response. We firmly believe that it is possible to develop algorithms that, taking as their inputs a model of the system (e.g., its code in RENDL), a description of the intrusion, and an actuation policy established by the system administrator, automatically obtain the set of actions needed to stop the offensive or, at least, to reduce its consequences. Although this work has been eminently theoretical, subsequent research effort could be oriented toward implementation issues of NSDF in a real environment where, regardless of the theoretical contributions, efficiency and real applicability are factors of crucial importance.

Acknowledgements

This work has been partially supported by Spanish MECD under National Program PNFPU (reference AP2001-3805).

Appendix A. RENDL language specification

This appendix describes the syntax and semantics of a simple language, termed RENDL (*RElational Network Description Language*), derived from the framework presented along this paper. The main objective of RENDL is to allow the description of a network system. Simply put, a description is given by an extensive formulation of the system state in terms of the entities which comprise it, and the relationships established among them. Therefore, a description of its state consists of a list of the entities and a list of the relationships. As was stated along Section 2.1, both entities and relationships are described by means of a list of pairs (attribute, value). While certain attributes are mandatory, like the identification or the operative state, each entity is allowed to have its own specific attributes.

Although the previous explanations provide the basis for completely describing a system within our framework, from a practical point of view it could be useful to have a mechanism for defining "skeleton" entities/relationships. Thus, we can define an object (entity or relationship) and assign to it default values. Subsequently, an object of that type can be created, in such a way that defined attributes maintain their default value if they are not modified. Having such a facility reduces considerably the amount of code that is needed to describe a system.

Within the previous context, the syntax of RENDL in BNF is provided in what follows.

```
<description>     ::= 'SYSTEM DESCRIPTION' <name> [<gen-defs>]
                      'BEGIN' <system-def> 'END'
<gen-defs>        ::= [<entity-type> '{' <e-attr-list> '}']
                      ['relationship''{' <r-attr-list> '}']
<system-def>      ::= <system>(<relationship>)*
<system>          ::= <description> |
                      (<entity>)+
<entity>          ::= 'define' <entity-type> <name> '{' <e-attr-list> '}'
<entity-type>     ::= 'ph_entity' | 'mac_entity' | 'ip_entity' | 'tp_entity' | 'ap_entity'
<e-attr-list>     ::= 'id' '=' <value> ';'
```

```
                          'nl' '=' <value> ';'
                          'as' '=' ('permitted' | 'not-permitted') ';'
                          'os' '=' ('operative' | 'not-operative') ';'
                          (<name> '=' <value> ';')*
<relationship>      ::=   'define' 'relationship' <name> '{' <r-attr-list> '}'
<r-attr-list>       ::=   'id' '=' <value> ';'
                          'ss' '=' <value> ';'
                          'ts' '=' <value> ';'
                          'as' '=' ('permitted' | 'not-permitted') ';'
                          'os' '=' ('operative' | 'not-operative') ';'
                          'tr' '=' ('vertical' | 'horizontal') ';'
                          're' '=' <boolean> ';'
                          'wr' '=' <boolean> ';'
                          'ex' '=' <boolean> ';'
                          (<name> '=' <value> ';')*
<boolean>           ::=   0 | 1
<name>              ::=   <string>
<string>            ::=   ['a'–'z' 'A'–'Z' '.' <digit>]+
<value>             ::=   <string> | <number>
<number>            ::=   <digit> | (<digit> '.' <digit>)
<digit>             ::=   ['0'–'9'] | (['1'–'9'] ['0'–'9']+)
<comment>           ::=   '#' <string>
```

Appendix B. An example of network description using RENDL

This appendix provides an example of application of RENDL. The following code contains a description of the system example illustrated in Fig. 5 and whose description within NSDF was given in Tables 1–4, and through Fig. 6. The code is briefly commented with the aim of helping to understand it. Previously, in order to facilitate the understanding of its structure, Fig. B.1 illustrates the general elements and sections that compose the code.

```
#######################################################################
# Description of a simple network system using RENDL
#######################################################################
SYSTEM DESCRIPTION SimpleNetwork

#========================
# Definitions of Generic Entities & Relationships
#========================
ph_entity {                          # Generic physical entity
        id = 0;                      # ------------------------------
        nl = 1;                      # layer 1
        as = "permitted";            # default value
        os = "operative";            # default value
        tp = "interface";            # default NIC
        sb = "rj-45";                # default RJ-45
        sp = 100;                    # default 100 Mbps
}
```

```
mac_entity {                            # Generic MAC entity
        id = 0;                         # -------------------------
        nl = 2;                         # layer 2
        as = "permitted";               # default value
        os = "operative";               # default value
        tp = "IEEE 802.3";              # type: default IEEE 802.3 access
        ad = "";                        # address
        mt = 1500;                      # mtu: default 1500 bytes
        mo = "non-promiscuous";         # mode: default non-promiscuous
}
ip_entity {                             # Generic IP entity
        id = 0;                         # ---------------------
        nl = 3;                         # layer 3
        as = "permitted";               # default value
        os = "operative";               # default value
        ip,ma,br = "";                  # IP, mask, broadcast
}
tp_entity {                             # Generic transport entity
        id = 0;                         # ------------------------------
        nl = 4;                         # layer 4
        as = "permitted";               # default value
        os = "operative";               # default value
        tp = "tcp";                     # type: default TCP
        st = "passive";                 # state: default server
        nu = 0;                         # number
}
ap_entity {                             # Generic application entity
        id = 0;                         # ---------------------------------
        nl = 5;                         # layer 5
        as = "permitted";               # default value
        os = "operative";               # default value
        ta,na = "";                     # type of access, resource name
}
relationship {                          # Generic relationship
        id,se,ts = 1;                   # ------------------------------
        as = "permitted";               # default value
        os = "operative";               # default value
        tr = "vertical";                # default support relationship
        re,wr,ex = 0;                   # all permissions disabled
}
#========================
# End of Definitions of Generic Entities & Relationships
#========================

BEGIN            # Begin of SimpleNetwork description

#==============
# Definition of System 'Host A'
#==============
SYSTEM DESCRIPTION HostA
BEGIN

define ph_entity e1ha {                      # physical entity
        id = 1;
}
define mac_entity e2ha {                     # MAC entity
        id = 2;
        ad = "00:50:DA:B9:63:AA";            # address
}
define ip_entity e3ha {                      # IP entity
        id = 3;
        ip = "150.214.60.223";               # IP
        ma = "255.255.255.0";                # mask
        br = "150.214.60.255";               # broadcast
}
define tp_entity e4ha {                      # transport entity
        id = 4;
        nu = 80;                             # number
}
define ap_entity e5ha {                      # application entity
        id = 5;
        ta = "http";                         # type of access
        na = "/wwwroot";                     # resource name
}

define relationship r1ha {                   # relationship between physical & MAC
        id = 1;
        ss = 1;
        ts = 2;
}
define relationship r2ha {                   # relationship between MAC & IP
        id = 2;
        ss = 2;
        ts = 3;
}
define relationship r3ha {                   # relationship between IP & TCP
        id = 3;
        ss = 3;
        ts = 4;
}
define relationship r4ha {                   # relationship between TCP & application
        id = 4;
        ss = 4;
        ts = 5;
}

END
```

```
#= = = = = = = = = = = = = = = = =
# End of Definition of System 'Host A'
#= = = = = = = = = = = = = = = = =

#= = = = = = = = = = = = = = = = = = = = = = = = = = = = =
# Definition of System 'Router R'
# Router R is composed of two subsystems, one in each network
#= = = = = = = = = = = = = = = = = = = = = = = = = = = = =
SYSTEM DESCRIPTION RouterR
BEGIN

#----------------------------------
# Definition of System 'RouterRInterface1'
#-----------------------------------
SYSTEM DESCRIPTION RouterRInterface1
BEGIN

define ph_entity e11r {                  # physical entity
        id = 6;
}
define mac_entity e21r {                 # MAC entity
        id = 7;
        ad = "00:50:DD:A7:45:30";        # address
}
define ip_entity e31r {                  # IP entity
        id = 8;
        ip = "150.214.60.223";           # IP
        ma = "255.255.255.0";            # mask
        br = "150.214.60.255";           # broadcast
}
# vertical relationships among entities in RouterRInterface1
define relationship r1r {                # relationship between physical1 & MAC1
        id = 5;
        ss = 6;
        ts = 7;
}
define relationship r2r {                # relationship between MAC1 & IP1
        id = 6;
        ss = 7;
        ts = 8;
}
END
#----------------------------------------
# End of Definition of System 'RouterRInterface1'
#----------------------------------------

#----------------------------------------
# Definition of System 'RouterRInterface2'
#----------------------------------------
```

```
SYSTEM DESCRIPTION RouterRInterface2
BEGIN

define ph_entity e12r {                  # physical entity
        id = 9;
}
define mac_entity e22r {                 # MAC entity
        id = 10;
        ad = "00:52:AD:B5:3F:F0";        # address
}
define ip_entity e32r {                  # IP entity
        id = 11;
        ip = "192.168.1.1";              # IP
        ma = "255.255.255.0";            # mask
        br = "192.168.1.255";            # broadcast
}

# vertical relationships among entities in RouterRInterface2

define relationship r3r {                # relationship between physical2 & MAC2
        id = 7;
        ss = 9;
        ts = 10;
}
define relationship r4r {                # relationship between MAC2 & IP2
        id = 8;
        ss = 10;
        ts = 11;
}
END
#------------------------------------------
# End of Definition of System 'RouterRInterface2'
#------------------------------------------

# In addition to subsystems RouterRInterface1 and RouterRInterface2, we have
# to define the horizontal relationship between IP entities of both subsystems in
# order to completely define system RouterR.

define relationship r5r {                # relationship between IP1 & IP2
        id = 9;
        ss = 8;
        ts = 11;
        tr = "horizontal";
}

END

#==================
# End of Definition of System 'RouterR'
#==================
#= = = = = = = = = = = = = = = = = = = = =
# Definition of System 'TransmissionMedium'
#= = = = = = = = = = = = = = = = = = = = = =
SYSTEM DESCRIPTION TransmissionMedium
BEGIN

# Entities in the media (wires)
define ph_entity e1c{                         # physical entity
        id = 12;
        tp = "link";                          # type
        sb = "multicast";                     # subtype
}
define ph_entity e2c {                        # physical entity
        id = 13;
        tp = "link";                          # type
        sb = "multicast";                     # subtype
}

END

#= = = = = = = = = = = = = = = = = = = = = = =
# End of Definition of System 'TransmissionMedium'
#= = = = = = = = = = = = = = = = = = = = = = =

#= = = = = = = = = = = = = = = = = = = = = = =
# Definition of System 'Host B'
#= = = = = = = = = = = = = = = = = = = = = = =
SYSTEM DESCRIPTION HostB
BEGIN

# Entities and relationships at host B
define ph_entity e1hb {                       # physical entity
        id = 14;
}
define mac_entity e2hb {
        id = 15;                              #MAC entity
        ad = "01:72:AF:96:33:FF";             #address
}
define ip_entity e3hb {                       #IP entity
        id = 16;
        ip = "192.168.1.2";                   #IP
        ma = "255.255.255.0";                 #mask
        br = "192.168.1.255";                 #broadcast
}
define tp_entity e4hb {                       #transport entity
        id = 17;
        st = "active";                        #client
     nu = 1945;                               #port
  }
```

```
define ap_entity e5hb {                #application entity
   id = 18;
   ta = "http";                        #type of access
   na = "/wwwroot/index.html";         #resource name
}
define relationship r1hb{              #relationship between physical & MAC
   id = 13;
   ss = 14;
   ts = 15;
}
define relationship r2hb{              #relationship between MAC & IP
   id = 14;
   ss = 15;
   ts = 16;
}
define relationship r3hb{              #relationship between IP & TCP
   id = 15;
   ss = 16;
   ts = 17;
}
define relationship r4hb {0            #relationship between TCP & application
   id = 16;
   ss = 17;
   ts = 18;
}
END

#= = = = = = = = = = = = = = = = = = =
# End of Definition of System 'Host B'
#= = = = = = = = = = = = = = = = = = =

#Finally, accessibility relationships among host A, host B, router R and media
define relationship r1s {               #relationship between phys_hostA & medium1
   id = 10;
   ss = 1;
   ts = 12;
   tr = "horizontal";                   # accessibility
}
define relationship r2s {               #relationship between phys_R1 & medium1
   id = 11;
   ss = 6;
   ts = 12;
   tr = "horizontal";                   # accessibility
}
define relationship r3s {               #relationship between MAC_hostA & MAC_R1
   id = 18;
   ss = 2;
   ts = 7;
   tr = “horizontal”;
}
define relationship r4s {                 #relationship between IP_hostA & IP_R1
   id = 19;
   ss = 3;
   ts = 8;
   tr = “horizontal”;
}
define relationship r5s {                 #relationship between phys_R2 & medium2
   id = 12;
   ss = 9;
   ts = 13;
   tr = “horizontal”;                     # accessibility
}
define relationship r6s{                  #relationship between phys_hostB & medium2
   id = 17;
   ss = 14;
   ts = 13;
}
define relationship r7s {                 #relationship between MAC_hostB & MAC_R2
   id = 20;
   ss = 10;
   ts = 15;
   tr = “horizontal”;
}
define relationship r8s {                #relationship between IP_hostB & IP_R2
   id = 21;
   ss = 11;
   ts = 16;
   tr = “horizontal”;
}
define relationship r9s{                 #relationship between TCP_hostA & TCP_hostB
   id = 22;
   ss = 4;
   ts = 17;
   tr = “horizontal”;
}
define relationship r10s {               #relationship between app_hostA & app_hostB
   id = 23;
   ss = 5;
   ts = 18;
   tr = “horizontal”;
}
END
#===================
# End of Definition of System ‘SimpleNetwork’
#===================
```

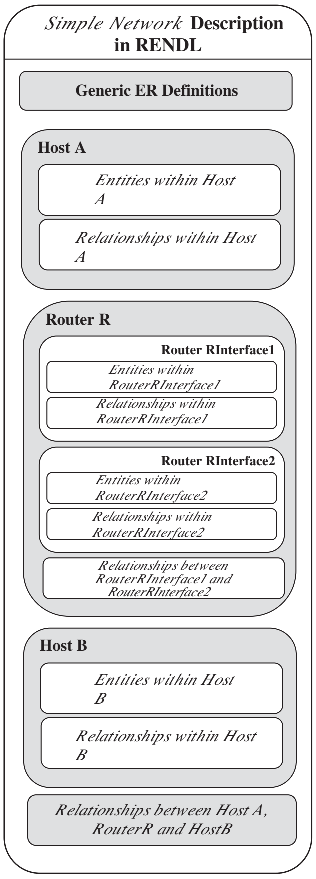
Simple Network Description in RENDL
Generic ER Definitions
Host A
Entities within Host A
Relationships within Host A
Router R
Router RInterface1
Entities within RouterRInterface1
Relationships within RouterRInterface1
Router RInterface2
Entities within RouterRInterface2
Relationships within RouterRInterface2
Relationships between RouterRInterface1 and RouterRInterface2
Host B
Entities within Host B
Relationships within Host B
Relationships between Host A, RouterR and HostB

Fig. B.1. Schematic structure of the RENDL code for the "SimpleNetwork" example.

References

[1] E.G. Amoroso, Fundamentals of Computer Security Technology, Prentice-Hall, Englewood Cliffs, NJ, USA, 1994.

[2] E.G. Amoroso, Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Traps, Trace Back and Response, Second Printing, Intrusion.Net Books, NJ, USA, June 1999, pp. 100–105 (Chapter 4).

[3] T. Aslam, A taxonomy of security faults in the unix operating system, Master's Thesis, Purdue University, West Lafayette, Indiana, USA, August, 1995.

[4] B. Beizer, Software Testing Techniques, second ed., Van Nostrand Reinhold, New York, 1990.

[5] C.A. Carver, U.W. Pooch, An intrusion response taxonomy and its role in automatic intrusion response, in: Proceedings of the 2000 IEEE Workshop on Information Assurance and Security, IEEE Computer Society Press, West Point, NY, USA, 2000.

[6] CERT® Coordination Center, CERT/CC Statistics for 1988 through 2002. Available from <http://www.cert.org/stats>.

[7] F.B. Cohen, Protection and Security on the Information Superhighway, Wiley, New York, USA, 1995.

[8] E.A. Fisch, Intrusion damage control and assessment: a taxonomy and implementation of automated responses to intrusive behaviour, Ph.D. Dissertation, Texas A&M University, College Station, TX, USA, 1996.

[9] C.B. Hogan, Protection imperfect: The security of some computing environments, Operating Systems Review 22 (3) (1988) 7–23.

[10] J.D. Howard, An analysis of security incidents on the Internet, 1989–1995, Ph.D. Dissertation, Department of Engineering and Public Policy, Carnegie Mellon University, Pittsburgh, PA, USA, April, 1997.

[11] J. Radatz (Ed.), IEEE, The IEEE Standard Dictionary of Electrical and Electronics Terms, sixth ed., Institute of Electrical and Electronics Engineers, Inc., NY, USA, 1996.

[12] C. Kiddle, R. simmonds, D.K. Wilson, B. Unger, ANML: a language for describing networks, in: Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Cincinnati, OH, USA, 15–18 August 2001, pp. 135–141.

[13] S. Kumar, Classification and detection of computer intrusions, Ph.D. Dissertation, Computer Sciences Department, Purdue University, Lafayette, IN, USA, August, 1995.

[14] C.E. Landwehr, A.R. Bull, J.P. McDermott, W.S. Choi, A taxonomy of computer program security flaws, ACM Computing Surveys 26 (3) (1994) 211–254.

[15] U. Lindqvist, E. Jonsson, How to systematically classify computer security intrusions, in: Proceedings of the 1997 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos, CA, USA, 1997, pp. 154–163.

[16] B. Mukherjee, L.T. Heberlein, K.N. Levitt, Network intrusion detection, IEEE Networks (May/June 1994) 26–41.

[17] P.G. Neumann, D.B. Parker, A summary of computer misuse techniques, in: Proceedings of the 12th National Computer Security Conference, Baltimore, MD, USA, 10–13 October 1989, pp. 396–407.

[18] A. Ogielski, Domain Modeling Language (DML) Reference Manual, 1999. Available from <http://www.ssfnet.org/SSFdocs/dmlReference.html>.

[19] T. Olovsson, Practical experimentation as a tool for vulnerability analysis and security evaluation, Ph.D. thesis, School of Electrical and Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, 1995.

[20] T.H. Ptacek, T.N. Newsham, Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection, 1998. Available from <http://www.securityfocus.com/data/library/ids.ps>.

[21] D. Russell, G.T. Gangemi, Computer Security Basis, O'Reilly, Sebastopol, CA, USA, 1991.

[22] R. Snodgrass, A relational approach to monitoring complex systems, ACM Transactions on Computer Systems 6 (2) (1988) 157–196.

Juan M. Estévez-Tapiador is a Ph.D. candidate in the Department of Electronics and Computer Technology of the University of Granada, Spain. He received an MS in Computer Sciences and Engineering from the University of Granada, where he obtained the "Best Student" Academic Award.

Currently he is holding a Ph.D. grant from the Spanish Ministry of Education. Since 2000, he is a member of the "Research Group on Signals, Telematics and Communications" of the University of Granada, where he has been involved in several public and private research projects. His research is focused on computer and network security, especially in intrusion detection and response systems, new security paradigms and group communications security.

Pedro García-Teodoro received his B.Sc. in Physics (Electronics speciality) from the University of Granada, Spain, in 1989. This same year he was granted by "Fujitsu España", and during 1990 by "IBM España".

Since 1989 he is Associate Professor in the Department of Electronics and Computer Technology of the University of Granada, and member of the "Research Group on Signal, Telematics and Communications" of this University. His initial research interest was concerned with speech technologies, especially automatic recognition, field in which he developed his Ph.D. Thesis in 1996. From then, his profile has derived to that of computer networks, and although he has done some works in telematics applications and e-learning systems, his main current research line is centred in computer and network security.

Jesús E. Díaz-Verdejo is Associate Professor in the Department of Electronics and Computer Technology of the University of Granada (Spain). He received his B.Sc. in Physics (Electronics speciality) from the University of Granada in 1989 and has held a Ph.D. grant from Spanish Government. Since 1990 is member of the "Research Group on Signal, Telematics and Communications" of the University of Granada. This year he became Assistant Professor at this University. In 1995 he obtained a Ph.D. degree in Physics. His initial research interest was related with speech technologies, especially automatic speech recognition. Currently he is working in computer networks, mainly in computer and network security, although he has developed some work in telematics applications and e-learning systems.