CHAPTER 2

REUSABILITY TECHNIQUES FOR BUILDING A SIMULATION ENVIRONMENT AND MODELING COMMUNICATION SYSTEMS

Atika Cohen and Radouane Mrabet

2.1. INTRODUCTION

The dynamic expansion of communication networks has grown considerably during the past two decades to meet the increasing demand of sophisticated users. The widespread use of LANs and the advent of new technologies such as ATM and frame relay are creating new problems for both managers and designers of telecommunication networks. Indeed, the design, analysis, and optimization of performance of such systems is a nontrivial task.

Nowadays, simulation plays an important role in computer-aided analysis, design, and management of communication networks. In fact, simulation technology is maturing and has been successfully applied during the design, development, and operational phases. It constitutes the only possible way to provide the network engineer with detailed information, when he has to decide regarding performance.

"Simulate before you buy or build" is becoming the norm, particularly for the OSISIM (Open Systems Integrated Simulator) project. This four-year research project was initiated by the "Université Libre de Bruxelles", represented by "Service Télématique et Communication", and by SAIT Systems, a Belgian company specialized into radio and satellite communication. The two main objectives of this project are : i) to set up an environment to model and simulate communication systems, and ii) to

define a methodology to model, in a generic way, mechanisms, protocols, and services related to communication systems.

One of the most significant criticisms on traditional simulation modeling has been the lack of reusability. However, the cost of using simulation technology can be reduced through the extensive application of model reusability. In addition, reusability is widely believed to be a key to improving development productivity and quality[1,2]. Therefore, reusability has been the key-word along the development of this work.

We address the question of reusability at three levels[3]: 1) reusability for creating new software systems from developed and tested software rather than from scratch, 2) reusability for creating complex models from building blocks already modeled, and 3) reusability for composing common functional elements to create building block models.

The first level of reusability, which can be considered as a coarse grain of reusability, is applied extensively to build an environment for modeling and simulation. As we will see later on, a prototype of this environment is based on four existing packages which have proved their usefulness in their respective field.

Another approach to reusability is to create components expressed in a simulation language and to group them into a library. Thus, a library can be defined as a database of reusable components. The issues which have to be considered when designing such a library are: what is the granularity and domain of application of the library?, how are components created, inserted, and maintained?, which relations among components may be expressed?, and what kind of knowledge is needed to build composite components from the library, while leaving the components unchanged in the course of their reuse?

This type of reusability is adopted to define the second and third levels of reusability. The main difference between the two levels is the granularity of components. For the second level, components are network elements, such as buses, rings, protocols, traffic sources, bridges, satellite repeaters, etc. While, the third level of reusability, which is considered as the fine grain of reusability, considers the functionalities of a protocol as reusable units, such as flow control, error recovery, segmenting, and rate control functions.

This chapter will address these three reusability levels in the three following sections. Section two focuses on AMS (Atelier for Modeling and Simulation), which is the simulation environment developed in the context

of the OSISIM project. Section three describes the communication network model unit, called DBM for Detailed Basic Model. Section four describes the protocol entity modeling unit which is the function. A conclusion is drawn in the last section.

2.2. SIMULATION ENVIRONMENT : AMS

We apply the first level of reusability to design a simulation environment for modeling and simulating communication systems. This environment is called AMS for Atelier for Modeling and Simulation[4,5], and it is designed in a modular fashion to allow the integration of several existing software.

AMS mainly differs from existing tools like OPNET[6] (Optimized Network Engineering Tool), TOPNET [7] (Tool for the Object oriented, Petri net based Network Evaluation and Test), and BONeS[8] (Block Oriented Network Simulator), in the formal methods and in the way it is used to describe the internal structure of the modeled components. OPNET is based on a graphical description of an extended finite state machine, and on the C language. With TOPNET, a modeled component is described by a class of timed Petri nets named PROT networks. BONeS has developed a block oriented paradigm. The blocks are graphically assembled and primitives written in C are at the lowest level of abstraction.

Unlike these approaches, ours is based on queueing networks. Indeed, queueing network models have come into widespread use as a modeling paradigm for deriving analytical as well as simulation based performance measures. They are especially effective in modeling computer communication systems, including point-to-point communication, broadcast systems, distributed multiple access systems, etc. QNAP2[9] (Queueing Network Analysis Package 2) has been selected to be the modeling and simulation language. It is a package for describing, handling and solving large and complex discrete event flow systems. It contains an object oriented specification language which is used for the description of the models and the control of their resolution.

In order to provide a wide range of features that facilitate modeling of communications networks, AMS includes on one side a library of basic models from which the end-user can construct a large number of transmission systems and networks, and on the other side several tools to edit the architecture, to simulate and to present results.

The two following subsections are organized as follows. The former describes in more details the functional structure of AMS, while the later presents the prototype we developed and which is based on this structure.

2.2.1 Functional Structure of AMS

AMS is designed in a modular manner. The main modules constituting AMS are structured in four phases, each of them is handled by one or several processes. These phases are the editing phase, ADL (Architecture Description Language) phase, simulation phase and presentation phase. Figure 2.1 illustrates this structure. Furthermore, the atelier is based on the library of basic models constructed in a very modular fashion to allow more flexibility.

Editing Phase: In this phase, the end-user constructs graphically a communication system using the basic models from the AMS library. The editing phase is managed by a dedicated process called the graphical editor which corresponds to the world view of the network designer, with icons that represent rings, buses, bridges, protocols, etc. Several instances of basic models can be created and connected by links for a specific architecture. The editor has the ability to provide substantial user interaction for either simple parameter changes or major reconfigurations of systems, and to permit the provision of hierarchical modeling capability for extremely large and complex models. On the other hand, an end-user without graphical capabilities has to describe his architecture directly in the ADL language.

ADL Phase: This phase mainly consists of translating the description written in ADL language into a QNAP2 simulation language. In addition, the aim of this phase is to fulfill several objectives mainly the definition of a clear border between the editing phase and the simulation phase. This separation will enforce the independence between the interactive part of the atelier represented by the editor and the computational part represented mainly by the simulator. This phase permits the design of the editor separately from the design of the computational part of the atelier, so that modifications in either tend not to cause changes in the other.

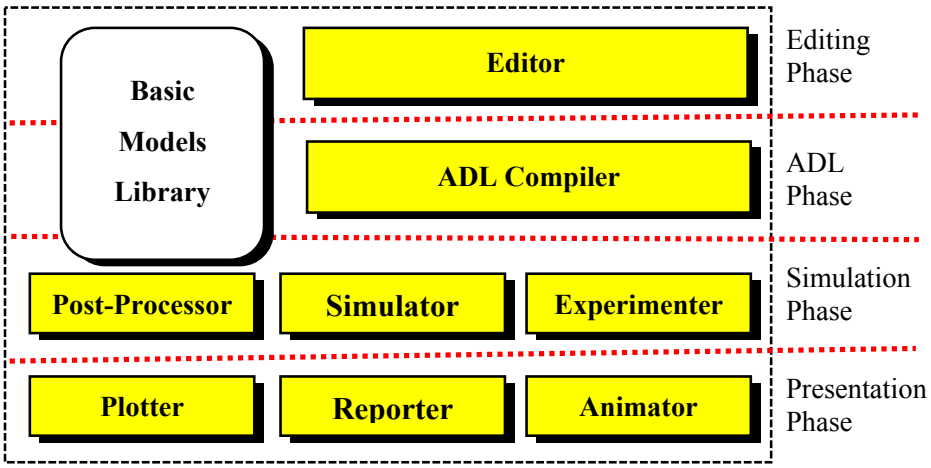

FIGURE 2.1 : AMS STRUCTURE

Simulation Phase: The main objective of this phase is to assess the performance of a previously edited system. This phase is handled by three processes which are the experimenter, the simulator and the post-processor. The experimenter goal is to help formalizing a proper experimental design to obtain the maximum information with the minimum number of experiments. The simulator process compiles and executes the QNAP2 code generated by the ADL process. The execution depends on the experiments defined by the end-user. At the end of the execution, all the desired rough results are produced. The recorded results of the simulation can be statistically analyzed with the post-processor, so that, aggregate parameters of interest can be reported.

Presentation Phase: This phase is meant to allow different presentations of the simulation results. It offers the end-user the possibility of supervising the simulation by visualizing its executions. In other words, it offers a high level animation showing, for instance, messages passing through the simulated architecture. This phase is handled by three processes which are the plotter, the animator and the reporter.

Library of Basic Models : The core of the AMS is the library of basic models which includes standard networks such as popular LAN technologies (Ethernet, Token Ring, FDDI, etc.), WANs (X25, TCP/IP), satellites (TDMA, FDMA, etc.), radio networks, and special network components like routers, bridges and gateways. Each basic model

corresponds to a communication entity; it is studied, verified, and validated separately. Depending on the phase where a basic model is used, it is represented by an icon, an ADL object, or by a QNAP2 object.

2.2.2 AMS Prototype

The first level of reusability is extensively applied. Indeed, a prototype of AMS is developed based on four existing and proven packages except the ADL language which is developed specially for the atelier. These packages are : QNAP2, GSS[10] (Graphical Support System), S-PLUS[11] and MODLINE[12].

Figure 2.2 shows the available tools in the prototype and on which packages they are based. The simulator is based on QNAP2, the post-processor is based on S-PLUS. The graphical editor and the animator are based on GSS which provides generic graph edition facilities to make the development of graphical performance modeling tools easier. The textual editing of an architecture can be done using ADL. The Reporter, Plotter, and Experimenter processes are based on MODLINE, a modeling environment conceived to assist in all stages of the performance analysis. On the other hand, AMS integrates all these packages through MODLINE which provides its graphical user's interface to the atelier. A prototype of the atelier has been developed on a SUN machine running the SunOs operating system. It is developed in C language. At present time, about 15000 lines have been coded.

2.3. COMMUNICATION NETWORK MODELING UNIT : DBM

So far, we said that the library is composed of basic models. More precisely, each basic model is to be detailed so as to reflect its exact behavior, and in the following it will be called DBM for Detailed Basic Model. We applied the second level of reusability, and DBM will be considered as a reusable unit for modeling communication network components. Accordingly, it has to implement, as precisely as possible, the operations performed by the target component, the component which is modeled by means of a DBM.

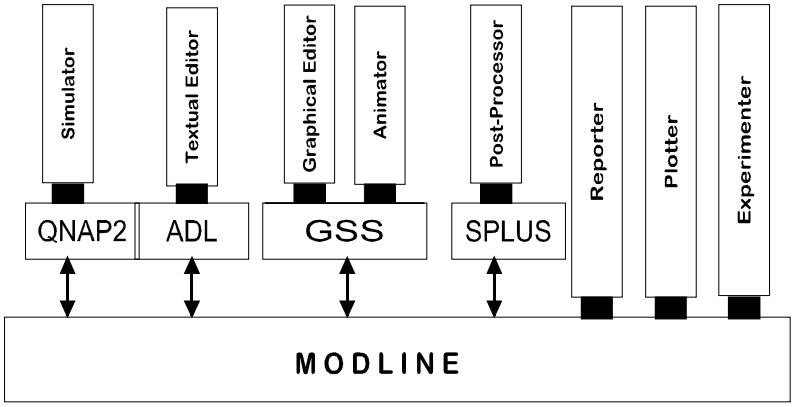

FIGURE 2.2 : AMS PROTOTYPE

A DBM can be used either as a component of a communication network or as a component of a generic model (for example complex network component). Thus, a DBM should be constructed in a very modular fashion, in order to achieve easily and efficiently this high degree of composition between DBMs.

A DBM is composed of three blocks : the Behavior Engine Block (BEB), the Interfaces Block (IB) and the Measurements Block (MeB). The structure is designed to meet the following fundamental objectives :

- Each DBM is self-contained, namely the behavior of the target component is modeled inside the DBM, and all the interactions with the outside world take place through interfaces.
- We keep in mind that each DBM may be a component of a communication system. Hence, a DBM must be connected to other DBMs; connections are made via interfaces. Interfaces have another important role, which is to free the BEB of messages exchange between the DBM and the outside world. This intends that the BEB has to handle only the behavior of the target component and not the issue of how to structure the messages.
- The main objective of modeling is performance assessment. Hence, statistical results are required. They will be processed during the simulation phase. Each DBM is intended to offer one or several measurements, which must be meaningful to an end-user.

DBMs are designed as objects which can be instantiated several times. The instances are linked together so as to mimic a given network

architecture or to specialize a generic model. The connection between DBM instances have to comply with a set of constraints, in order to construct coherent systems which can operate correctly during their execution.

A DBM can be characterized by a number of parameters. The parameters are classified into two classes, the configuration parameter's class and the performance parameter's class. The parameters of the first class are transparent for the end-user, their values depend on the context where the DBM is instantiated and/or on some characteristics of the DBM. Namely, the values of these parameters are set according to the configuration of the network where the DBM is used. The parameters of the second class can be used by the end-user to assess the performance of the DBM.

The following sub-sections explain the internal structure of a DBM more thoroughly.

2.3.1 Behavior Engine Block

The behavior of a target component is modeled within the BEB of its associated DBM. The BEB is an open network of stations. Each station includes a queue with limited or unlimited capacity, and one or several servers. The network of stations is open because it receives from and/or sends to the outside, through the interfaces, different messages. The configuration of the network of stations and the services offered by each station are left to the responsibility of the modeler. The complexity of the network of stations heavily depends on the complexity of the target component.

2.3.2 Interfaces Block

The interfaces block is an important part of a DBM for several reasons. The modularity aspect of a DBM is reinforced by its presence. It allows the modeler to develop a DBM independently of any system of which it can be a component. Well-defined interfaces also promote the reuse of the global model. Finally, it frees the BEB of the task of message exchange with the outside world.

A DBM can have several interfaces. Their number can either be a fixed value known during the modeling phase or it can vary. In the later

case, the modeler can progressively add new interfaces to the DBM. A given interface can be instantiated one or several times if an instance of the current DBM is connected to one or several instances of a DBM through this given interface.

An interface is represented by means of two stations (figure 2.3), each station is composed of a queue and a server. The station Qin receives messages from the outside, which are meant to be sent later to the BEB.

Qout receives messages from the BEB to be sent later to another DBM, which is connected to it. The services inside Qout and Qin both depend on the type of DBM connected to the current DBM as well as the type of messages to be handled by the current DBM.

2.3.3 Measurements Block

The measurements block contains two types of measures. The first type reflects the behavior of a DBM, and it is associated with the BEB. The second type is associated with the interfaces, and it mainly shows the data flow entering and exiting the DBM.

The modeler of a DBM defines a list of measurements. The measurements must be meaningful to an end-user who is not specialized in the field of queueing networks. All the aspects related to this field are transparent. These measurements must be related to some metrics currently used in the field of communication systems, e.g. throughput, response delay, etc.

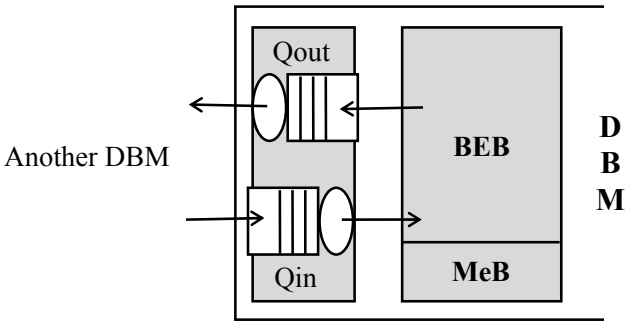

FIGURE 2.3 : A STANDARD INTERFACE

2.4. PROTOCOL ENTITY MODELING UNIT : FUNCTION

The fine grain unit of reusability is the concept of function. A function is defined in OSI RM[13], as part of the activity of a protocol entity, knowing that most protocol entities can be expressed in terms of functions. Based on this concept, we have developed a methodology[14,15] to model protocol entities as DBMs, especially, their behavior engine blocks. Presently, only the internal structure of a DBM has been defined and not the way to describe the BEB. This leaves to the modeler the hard task of its description.

Each function will be separately developed as a reusable unit, which will facilitate its implementation, verification, and if necessary its maintenance. Based on these reusable units, which interact in a simple and well-defined way, the modeler can rapidly build well-structured models of protocol entities. Thus, the proposed methodology will help the modeler to partly fill the gap between specifying and modeling protocol entities.

On the other hand, a high degree of service flexibility can be provided. Indeed, applications can use efficient communication subsystems tailored to their individual needs. Clearly, a protocol entity pattern will be composed of a sub-set of functions selected from a library called LoF, for Library of Functions. The selection is driven by the application's needs and the services offered by the underlying sub-networks.

This approach is pursued in many other projects, specially to design the new generation of high speed transport protocols for high speed communication systems[16,17,18,19,20].

2.4.1 Function Concept

The Library of functions (LoF) is a set of simple functions. LoF = $\{f_1, f_2, ..., f_n\}$ with $n \geq 2$. A simple function is defined as a function performing one task, provided this task is atomic, i.e. it does not need the cooperation of other functions to be achieved. A cooperation between functions is possible when all or part of the functions are asked to provide a given service. Henceforward, only the term "function" will be used instead of "simple function".

Three types of functions are considered : Prerequisite, Selected and Pulled functions. A function is defined as prerequisite, if and only if, it is

always present in a protocol entity pattern, whatever the service required. A "selected function" is a function which is chosen initially by the modeler to be present inside the pattern of a protocol entity. As a rule, a selected function has to be present inside the pattern of the protocol entity. A "pulled function" is a function which is chosen by the "pattern protocol determination process" and which is neither a prerequisite function nor a selected function. This type of function has to be present in a protocol entity pattern in order to make the pattern consistent.

Each function f_i can perform its task in different ways, which implies that it has to be associated with different algorithms (the term mechanism is also used). Let us denote :

- Δ_i : the set of algorithms associated with f_i ($\Delta_i \neq \varnothing$)
- Π_i : a sub-set of Δ_i including at least one algorithm ($\Pi_i \subseteq \Delta_i$ and $\Pi_i \neq \varnothing$)
- Furthermore, we write f_i/Π_i to denote that function f_i is associated with a sub-set of its algorithms. Π_i may contain only one algorithm A_{ij} (jth algorithm of f_i).

A function f_i/A_{ij} is modeled as a module with well-known boundaries represented, in our case, by sets of inputs and outputs. $Sin(f_i, A_{ij})$ is the set of the object attributes whose values can be used by a function f_i/A_{ij} during its execution. Clearly, the values of these object attributes may be read by the function f_i when the algorithm A_{ij} is executed. $Sin(f_i, A_{ij})$ is never empty, because when a function is triggered by an event, and has to perform a certain job, it has to know at least the type of the event and/or the context where this event occurs.

$Sout(f_i, A_{ij})$ is the set of the object attributes whose values are updated by f_i during the execution of the algorithm A_{ij}. As for $Sin(f_i, A_{ij})$, it may happen that not all the object attributes of $Sout(f_i, A_{ij})$ are updated. $Sout(f_i, A_{ij})$ is never empty.

The functions of LoF can be partitioned into non-empty sub-sets. The partitioning is based on the class of events processed by the protocol entity, which can be classified into essentially three classes. The first class is defined by the arrival of messages from the application layer, the layer which asks for a service. The second class is defined by the arrival of segments from the sub-network which is beneath the protocol entity layer, while the third class is defined by the expiration of timers. This partitioning will allow to define a partial order between the functions belonging to the same part, while it does not exclude the existence of relations between the functions belonging to different parts.

2.4.2 Relations Between Functions

A function may be related to the other functions by means of two graph types, precedence graphs and $\mathfrak{E}$-Graph. In the first type of graph, the partial order between functions is captured, while in $\mathfrak{E}$-graph, mutual presence of functions in a protocol is captured. Hereafter, a formal description of these two types of graphs.

1. Precedence Graph. A partial order between the functions belonging to the same part of LoF is defined as the acceptable order in which operations can be performed. The reason for dependencies could be viewed as some sort of shared information, manipulated by a function and required by another one. So, the necessary condition for function f_i/A_{ip} to precede function f_j/A_{jq} is : $Sout(f_i/A_{ip}) \cap Sin(f_j/A_{jq}) \neq \varnothing$. Two main types of precedence are defined : *strong precedence* and *weak precedence*.

- *Strong Precedence* : Function f_j/A_{jq} is preceded strongly by function f_i/Π_i ($i \neq j$) means that if f_j/A_{jq} has to be part of a protocol entity pattern then f_i also has to be part of this pattern with one algorithm belonging to Π_i (let it be A_{ip}). The statement "a function has to be part of a protocol entity pattern" is a general statement which means indirectly that a function is either a prerequisite, selected or pulled function.
- *Weak Precedence* : Let us assume that function f_j/A_{jq} has to be part of a protocol entity pattern. If f_j/A_{jq} is preceded by function f_i/Π_i ($i \neq j$) by means of a weak-arc, then f_j/A_{jq} will always be executed even if f_i will not be part of this pattern. In the case where f_i has to be part of the pattern then it has to be associated with an algorithm belonging to Π_i.

Definition : A precedence graph G_i, associated with a sub-set F_i, is a digraph. The vertices of G_i are the functions of F_i. The G_i arcs represent the precedence relation between the functions. An arc is either a "strong-arc" or a "weak-arc". Each arc is labeled. If it connects functions f_i and f_j, the label is denoted L_{ij}^{pq} (f_i/A_{ip} precedes f_j/A_{jq}). G_i has two special vertices, the "Begin" vertex and the "End" vertex. The former is not preceded by any other vertex, and the later does not precede any vertex.

2. $\mathfrak{E}$-Graph. An $\mathfrak{E}$-Graph is a graph which may have arcs and edges. A connector is either an arc or an edge. The vertices of an $\mathfrak{E}$-Graph are the functions of LoF. The two extremities of a connector have to belong to two

different parts of LoF. A connector linking two functions f_i and f_j is labeled either by the symbol Γ or the symbol Λ.

If f_i/Π_i is connected to f_j/Π_j by a Γ-labeled (respectively, Λ-labeled) connector and the function f_i has to be part of a protocol entity pattern with an algorithm belonging to Π_i then the function f_j has also to be part of this protocol entity pattern (respectively, the peer protocol entity pattern) associated with an algorithm belonging to Π_j. The opposite is not true in the case where the connector is an arc.

2.4.3 Protocol Entity Pattern

A protocol entity has to be modeled as a DBM to be added to the AMS Library. Therefore, the internal structure of a DBM has to be taken into account. Figure 2.4 shows the structure of a protocol entity and mainly the queueing network modeling the Behavior Engine Block. Actually, the BEB is composed of three stations and a specific process, TiM for Timers Manager, which is dedicated to the management of timers. The service of each station executes the selected functions of a given part.

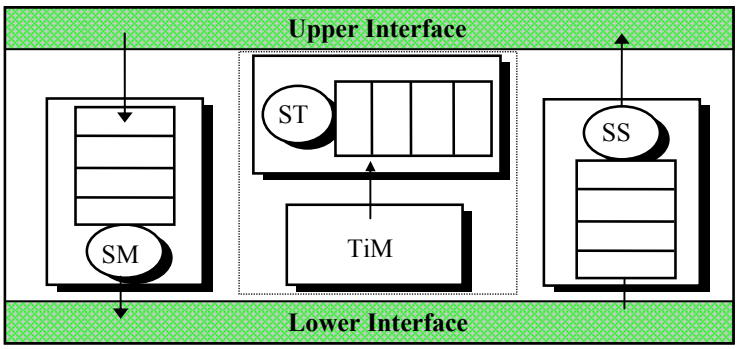

FIGURE 2.4 : BEB STRUCTURE OF A PROTOCOL ENTITY

As if the structure of a BEB is simple, it is efficient for two main reasons: i) the modeler can recognize in which station service a given function should be used, ii) the number of stations is limited to only three stations, thus the interactions between the stations are also limited, which leads to a reduction of the simulation time of the model.

Concerning the measurements, they are related to the function concept. A measurement will be seen as a hook on a function or a sub-set

of functions. A function can be associated with zero, one or more than one measurement(s). In the case where at least one measurement is associated with one function, this measurement will be used to assess the behavior of this function. If a measurement is associated with a group of functions, then it will be computed only if all the functions of the group are chosen to be part of the pattern of the protocol entity.

The pattern of a protocol entity will be tailored according to the application needs. So, building up a pattern of a protocol entity is driven by the required service. A given service can be seen as a set of functions which have to be implemented by a protocol entity. External constraints which generally come from the sub-network have to be taken into account by the modeler when he makes his selections. Indeed, these constraints either require to select additional function(s) or on the contrary, to cancel the pre-selected function(s).

We have developed a specific process, called PPDP for Protocol Pattern Determination Process. It is used to construct a pattern of a protocol entity from LoF which will provide a given service. PPDP uses the precedence graphs and the $\mathcal{E}$-graph defined for the available functions into LoF. PPDP constructs at the same time a pattern of a protocol entity and a pattern of its peer-protocol entity. In these two patterns, the prerequisite functions have to be present, as well as the selected functions, and maybe also other functions to give coherence to the assembled set of functions.

2.4.4 Example

XTP[21] is one of the most promising full-featured light-weight transfer protocols; it provides the functionalities of Network Layer and Transport Layer. XTP can provide a full range of services needed to support distributed systems. The features of XTP include rate control, selective retransmission, no-error mode, etc.

We model XTP using the function-based methodology described above. The functions which are modeled are stored in LoF and are partitioned into three parts : Fmsg, Fpack and Ftimer. Each part is associated with an event class, arrival of messages, arrival of segments or expiration of timers. Figure 2.5 shows the precedence graphs associated with Fmsg. Figure 2.6 depicts the $\mathcal{E}$-Graph, where only the connected

vertices are shown. We assume that each function is associated with only one algorithm.

LoF = {fragmentation, flow control, rate control, padding, numbering, retention, transfer DT, go-back-n on wtime, coalesce, un-padding, resequencing, selective retransmission, rtt estimation, transfer CNTL}

Fmsg = {fragmentation, flow control, rate control, padding, numbering, retention, transfer DT}

Ftimer = {go-back-n on wtime}

Fpack = {coalesce, un-padding, resequencing, selective retransmission, rtt estimation, transfer CNTL}.

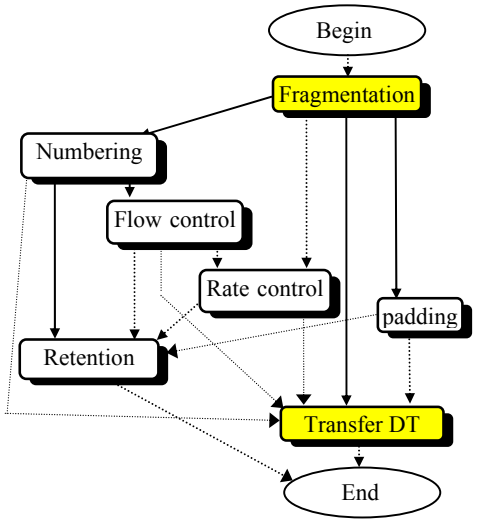

FIGURE 2.5 : PRECEDENCE GRAPH OF FMSG

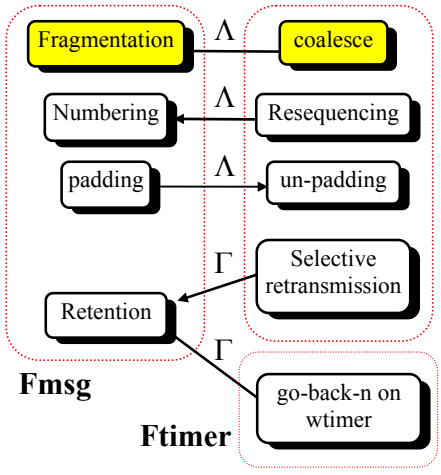

FIGURE 2.6 : THE $\mathcal{E}$ GRAPH

2.5. CONCLUSIONS AND FUTURE DIRECTIONS

This work has been accomplished in the context of the OSISIM project. The main contribution is the emphasis put on the reusability concept, on

one side, for designing a simulation environment, and on the other side, for defining two different levels of granularity for reusable network component libraries. In retrospect, one can clearly see the positive potential influence that reusability is having on the development of high quality software and models.

The design of our simulation environment, called AMS, was based on existing pieces of software, which proved their usefulness in their respective fields. In order to carry out this integration efficiently, a modular structure of the atelier was proposed.

The kernel of the AMS is its library of Detailed Basic Models (DBMs). Each DBM was designed in order to comply with the most important criterion which is reusability. Indeed, each DBM can be used in several network architectures and can be a component of generic and composite models. Section 3 was dedicated to the description of the internal structure of a DBM.

The most important contribution of this research is the definition of a methodology for modeling protocol entities as DBMs. We then tried to partly bridge the gap between specification and modeling. Section 4 presents this methodology which is based on the concept of function. Simple functions are modeled as reusable modules and stored into a library.

The Function Based Methodology was designed to help the modeler efficiently and rapidly build new protocols designed for the new generation of networks where several services can be provided. These new protocols can be dynamically tailored to the user's requirements.

The work achieved during this work can be enhanced in many directions. The major research direction which emerged during our reflection is to apply hybrid simulation modeling (HSM) techniques[22,23] in order to tackle the main disadvantage of simulation which is the slowness of the execution. The goal of HSM is to build models which are, on one side, more representative than pure analytical models and, on the other side, that lead to a substantial reduction of execution time with respect to pure simulation models. The HSM techniques were already used for particular situations but reusability was not taken into account.

In order to define more formally HSM, we refer to the classification introduced by Shanthikumar and Sargent[24], that distinguishes four classes of HSM by considering four different interaction ways between simulation and analytical models. Specifically, two of these

classes (I and II) include the combination over time of simulation and analytical solutions either in parallel or through a joint solution procedure. The other two classes (III and IV) consider either a pure analytical or simulation model of the total system and use, respectively, a simulation or an analytical solution to represent a portion of the system. In our case and for reusability purposes, the fourth class of HSM has to be studied more thoroughly. Actually, the total system can only be resolved by simulation with one or several parts modeled analytically.

2.6. REFERENCES

[1] W. TRACZ, Software Reuse: Emerging Technology (IEEE Computer Society Press, 1990).

[2] E.T. SAULNIER and B.J. BORTSCHELLER, "Simulation Model Reusability", IEEE Communications Magazine, 32, 64 (1994).

[3] R. MRABET, "Reusability and Hierarchical Simulation Modeling of Communication Systems for Performance Evaluation. Simulation Environment, Basic and Generic Models, Transfer Protocols." Ph.D. Thesis, Applied Sciences Faculty, Université Libre de Bruxelles, Brussels, Belgium, June 1995.

[4] A. COHEN and R. MRABET, "AMS : An Integrated Simulator for Open Systems", Proceedings of IEEE GLOBECOM'93 Conference, (Houston, Texas, 29 November - 2 December 1993), 656.

[5] COHEN and R. MRABET, "An environment for Modelling and Simulating Communication Systems, Application to a System Based on a Satellite Backbone", The International Journal of Satellite Communications, **13**, 147 (1995).

[6] F.H. DESBRANDES, S. BERTOLOTTI, and L. DUNAND, "OPNET 2.4: An Environment for Communication Network Modeling and Simulation", Proceedings of European Simulation Symposium, (Delft, The Netherlands, October 1993), 609.

[7] M. A. MARSAN, G. BALBO, G. BRUNO, and F. NERI, "TOPNET : A Tool for the Visual Simulation of Communication Networks", IEEE Journal on Selected Areas in Communications,

8, 1735 (1990).
[8] K. S. SHANMUGAN, V. S. FROST, and W. LARUE, "A block-Oriented Network Simulator", Simulation, 83 (1992).
[9] SIMULOG S.A. "QNAP2 User's Manual", Version 10, (1992).
[10] SIMULOG S.A. "GSS User's Guide", Version 4, (1992).
[11] MATHSOFT INC. STASCI, "S-PLUS User's Guide", Version 3.2, (1993).
[12] SIMULOG S.A "MODLINE User's Guide", Version 3.3, (1995).
[13] ISO International Standard 7498, "Information processing systems — Open Systems Interconnection : Basic Reference Model", (1984).
[14] A. COHEN and R. MRABET, "Function-based Methodology to Model Communication Protocols for Performance Evaluation", Proceedings of International Conference on Communication Systems, (Singapore, 14-18 November 1994), 90.
[15] A. COHEN and R. MRABET, "Modeling Function-Based Communication Protocols Entities for Performance Assessment, Application to XTP", Proceedings of Twelfth International Conference on Computer Communication, (Seoul, South Korea, 21-24 August 1995), 75.
[16] D.D. CLARK and D.L. TENNENHOUSE, "Architectural Considerations for a New Generation of Protocols", Computer Communications Review, **20**, 200 (1990).
[17] Z. HASS, "A Protocol Structure for High-Speed Communication over Broadband ISDN", IEEE Network Magazine, **5**, 64 (1991).
[18] M. ZITTERBART, "High-Speed Transport Components", IEEE Network Magazine, **5**, 54-63 (1991).
[19] D.C. FELDMEIER, "A Framework of the Architectural Concepts for High-Speed Communication Systems", IEEE Journal on Selected Areas in Communications, **11**, 480 (1993).
[20] D.C. SCHMIDT and T. SUDA, "Transport System Architecture Services for High-Performance Communications Systems", IEEE Journal on Selected Areas in Communications, **11**, 489 (1993).
[21] Protocol Engine Inc., Xpress Transfer Protocol, Version 4.0 (1995).
[22] V.S. FROST, W.WOOD LARUE, and K.S. SHANMUGAN, "Efficient Techniques for the Simulation of Computer Communications Networks", IEEE Journal on Selected Areas in

Communications, **6**, 146 (1988).

[23] S. BALSAMO, M. CAPPUCIO, L. DONATIELLO, and R. MIRANDOLA, "Some Remarks on Hybrid Simulation Methodology", Proceedings of Summer Computer Simulation Conference, (Calgary, Canada, 16-18 July 1990), 30.

[24] J.G. SHANTHIKUMAR and R.G. SARGENT, "A Unifying View of Hybrid Simulation/Analytic Models and Modeling", Operations Research, **31**, 1031 (1983).