Philippine Engineering Journal (1996) XVII (1): 19-32

THE GENERIC NETWORK MODELING LANGUAGE (GNM)

Erville D. Magtubo
Communications Engineering Division
Advanced Science and Technology Institute
Department of Science and Technology
Philippines

ABSTRACT

GNM is a language specifically designed to be the base language for the HERMES Project. This language provides network engineers a tool for describing and designing a network topology. With GNM, engineers can design their network in a graphical manner making it easier to be understood by other engineers. This paper discusses the basic rules in using the language.

INTRODUCTION

The design and development of computer networks has never been an easy task. From the drawing board to the actual implementation, computer networks may come out with a lot of problems. The efficiency, speed and accuracy of the network are just a few of the issues that the network designer has to take into account. These issues become more evident as the need for network systems, by almost any kind of business, grows.

To aid developers in their design of various computer networks, a number of development systems are available in the market. These systems, usually software, help the developer design, document and evaluate their network. It also cuts their total development time by a considerable amount making development not only efficient but also fast. **BONeS** and **OPNET** are examples of such development systems that are available in the market.

An essential part of these development systems is their language or modeling paradigm. These languages (from now on, language will not mean a programming language in the strict sense but some kind of modeling paradigm or behavior), usually following a certain methodology that the development system promote, help the user in the design and specification of the network. After the developer has specified the network through the language, the development system can now perform a number of task that will test, verify and sometimes simulate the network. These languages can vary from being purely text to something that is highly graphical. Basically the language serves as the mediator between the developer and the development system.

GNM is one such language or modeling paradigm. It is the language used by the **HERMES** software, a network modeling and simulation tool developed to cater for students, professors, researchers and for those who cannot afford the very expensive network development

system. The developer models the network using GNM then HERMES simulates it and provides the developer data from which he can evaluate the network that he designed.

This paper discusses the details of GNM. It will provide the developer the know how to design networks using GNM. After reading this paper one can expect to be a knowledgeable GNM "programmer".

One more note, GNM is not a language in the strictest sense. It can be thought of as a mere concept where further formalization can be done to make it a full pledged language.

GENERIC NETWORK MODELING LANGUAGE

GNM stands for Graphical Network Modeling. It is a graphical language that aids the network developer in his design. It aids the developer by providing him with a set of rules or guidelines to work by. And since it is graphical, the output is a top to bottom overview of the designed network.

Primarily, GNM is a language for the purpose of simulation. The network to be simulated can be modeled with GNM then an engine can read this GNM specification and simulate the model accordingly. To date, the only engine that can interpret or simulate a GNM specified network model is **HERMES.**

GNM is a relatively simple language. Since it caters to students (would be network developers) and educators, GNM was designed to be simple yet powerful enough to be able to model most of the well know network topologies and protocols. As much as possible, GNM models the network close to reality.

Among the seven layers of the OSI Protocol Hierarchy. GNM focuses on the three lower layers. Modeling the *Network, Data link*, and *Physical* layers of the network would be relatively easy. Although, modeling the other four upper layers is possible with GNM, it would be a bit complicated.

GNM is *object-oriented* and *hierarchical*. It treats the elements of the network as objects having certain properties and behavior. The objects are composed of several objects and certain behaviors forming a hierarchy of objects. It is also event-driven. Since networks are governed by events, GNM was designed to allow the developers define events from which elements of the network can respond to.

In defining the behavior of the network, GNM employs the *block-oriented* language adopted from BONeS. Here procedures or "codes" are treated as blocks in which these blocks are composed of a set of connected blocks forming a hierarchy of blocks. This way "coding" will be much simpler and code reusability will be promoted.

GNM STRUCTURE

A GNM network design is composed of four design levels. These design levels form the hierarchy that make up the network design. The following are the four levels:

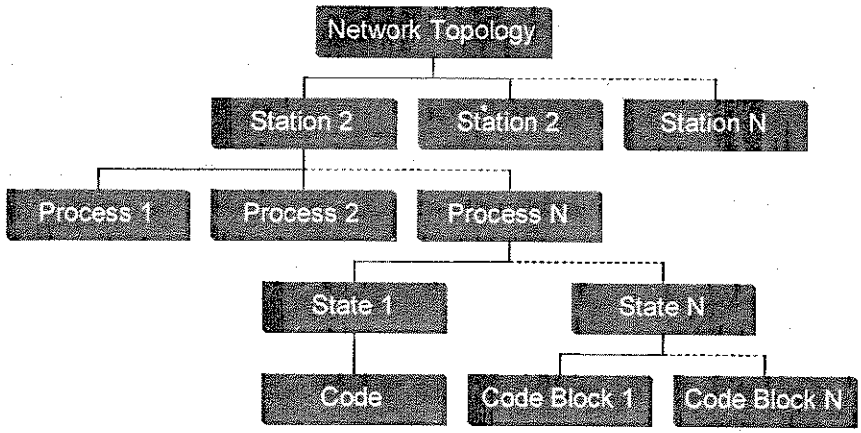


Figure 1. GNM Design Hierarchy

- Network - top level of the network design; the overall topology of the network is manifested on this level of design
- Node/Station - second level: contains the definitions of the nodes or stations that comprise the network
- Process - it is in this level where the behavior of each process in a station is defined: the actions of the stations in response to events are specified in this level
- Block - the underlying language of GNM; it is in this level where the procedures and actions are defined

At the lowest level, blocks are composed of several blocks connected together. In effect, blocks can be blown up to see the composition of that block. Blocks that can no longer be blown up are called *primitives*. These primitives where are no longer represented by a set of connected blocks but are defined using a certain language (in HERMES, the primitive code used is C).

As was mentioned, GNM is hierarchical. Its hierarchical structure can be viewed as a tree where the network level is the root and the primitives are the leaves.

With this hierarchical structure, it will be easier to visualize a network configuration or design. Each component of the network is treated as objects making the definition of its behavior much easier.

DATA IN GNM

An essential part of a network is its data. The data in network can vary from a simple control signal, or to a complex data packet. GNM allows the developer to define his data requirements. He is allowed to define almost any kind of data type and of any size. Although, varying implementations may have varying restrictions and limitations in their data handling.

GNM has no formal data declarations of types and variables like those that C and Pascal have. Depending on the implementation of GNM declaration of variables and data types may vary. For example in HERMES data types are defined graphically.

Although GNM does not have any formal declarations of types. GNM recognize most of the common data types in any programming language. The basic data types are as follows:

1. integers
2. integer vector
3. integer matrix
4. real
5. real vector
6. real matrix
7. string
8. event (can be implemented as string)
9. void

Of course, an implementation of GNM can have more than the basic data types above.

Aside from the basic data types, one can define structures composed of the basic ones. Most of the time, the data requirements of networks cannot be represented by a simple integer or array. GNM allows one to define data structures. This way complex data like packets, and node acknowledgments can be represented properly.

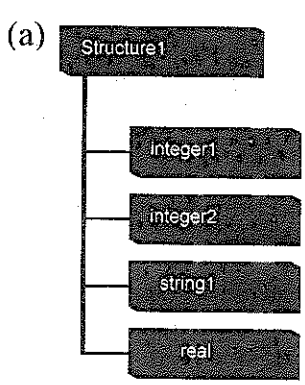
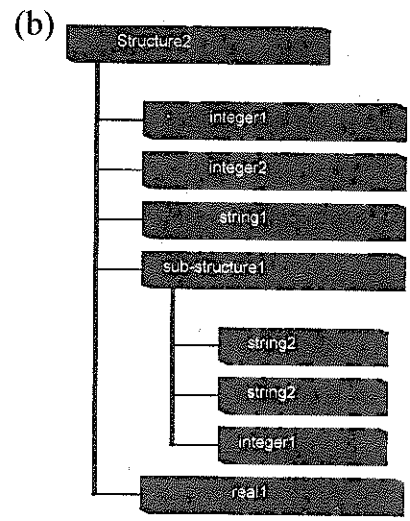


Figure 2. (a) A simple structure



(b) Structure within structure

There are two types of data in GNM

- Memory - variables that contain data can be defined with the use of *memory*. With the use of memories, one can record data and keep track of the information pertinent to any design level of GNM. Memories can also function as a communication bridge between objects.
- Parameters - like functions in a programming language (like C for example). one can define objects that require parameters. These parameters serve as input and output containers with which one can pass data down in the design hierarchy.

THE NETWORK GEOMETRY

One of the important issues in designing a network is its topology. The topology simply defines the connections from node to node, or even connections from one network to another.

Also, another concern in the network geometry is the medium. There are a wide range of media that can connect two nodes which has varying speeds and error tolerance. The table below shows some of the common media.

Medium	Signal Propagation Speed
sound in air	0.000001c
twisted pair	0.8c
coaxial cable	0.8c
radio waves	1c
optic fiber	0.85c

As was mentioned the GNM language has four levels of design:

1. Network level
2. Node/ Station level
3. Process level
4. Block level

THE NETWORK DESIGN LEVEL

There are a number of different network topologies that can be adopted for a specific network design. It can be as simple as full duplex link between two computers or as complicated as hierarchical switched network. These network topologies are modeled in the network design level.

The network design level of GNM is where the developer designs the network topology. It is in this level where he adds the nodes or stations of the network and links them accordingly. It is also in this level where the types of links are determined.

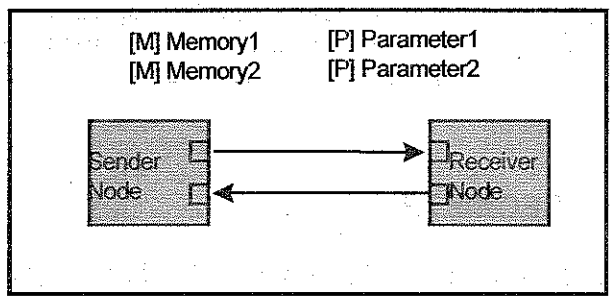


Figure 3. Memory and Parameter example

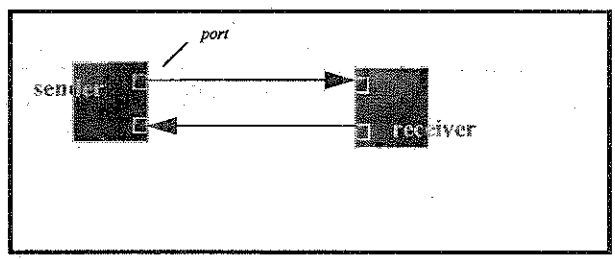


Figure 4. A network example

The active objects in the network design level are the **nodes** or **stations**. These stations may represent a computer node, a gateway or a file server. Also, these stations are the ones that generate the traffic in the network.

Stations have what we call **ports**. Ports are basically the connection of the stations to the other stations. The links are connected via these ports. If a station does not have any ports, then it cannot be connected to any station. All data sent and received pass through this ports. It serves as a window for the data to pass through going to stations connected to it. These ports can be thought of as a modem connected to a computer.

The example in Figure 4 is the classical sender-receiver topology. A sender node just sends a packet to a receiver node and then the receiver sends an acknowledgment. The links are implemented by two unidirectional links so there is no possibility of collision.

In the figure, the big blocks are the stations. As shown, there are two stations, the sender and the receiver station. The small squares are the ports and the arrow lines are the **links**. This links represent the connections and flow of data in the design.

STATION/NODE DESIGN LEVEL

Stations are the major entities in any network. Stations can be computers, workstations or routers. It can also be transmitters, receivers or network repeaters. These stations are the on-- responsible for creating traffic in the network.

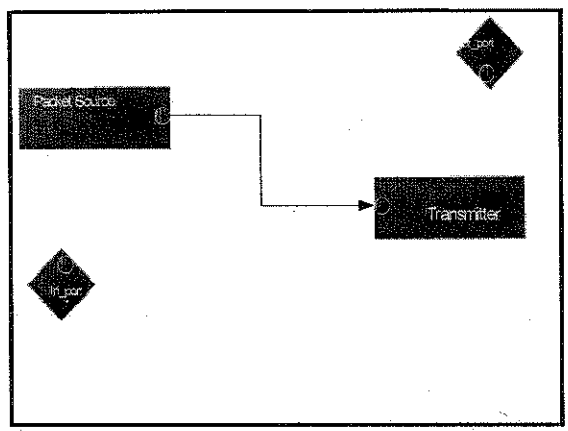


Figure 4.1. A node design example

In GNN, nodes are viewed as a group of cooperating processes. These processes define the overall behavior of the node in the network. It is in this design level that a node is defined.

As was mentioned in the above paragraph, nodes in GNM are merely a collection of cooperating processes. So, defining a node is simply adding the processes that will compose the overall behavior of the node, Queues, random generators, and math functions are just some of the processes that a node may need.

In Figure 4.1 the blocks represent the processes. Process generate events with which other processes respond to. The diamonds are the ports. Ports are also processes and are in charge of all the reception and transmission of data to other stations.

For processes to cooperate, it must be provided by a means to communicate. This communication will allow processes to pass data and synchronize with each other. GNM allows interposes communication. Processes communicate via shared memory or through the use of data ports.

Processes can be use memories to communicate. Since a defined memory in any design level is visible to any of the objects, processes can pass data through the memories or use it as synchronization control. But, the problem with common memories is that all the processes in the same node can access the memory location. To allow for a little protection **data ports** and **paths** are introduced.

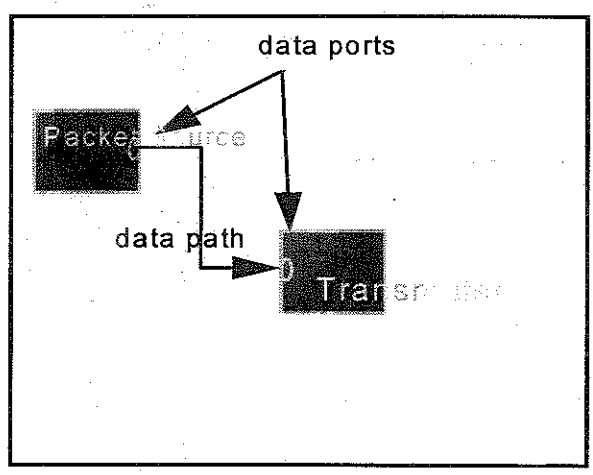


Figure 4.2. Data ports and paths

Data ports are simply memory variables where it is local only to the process who owns it. The memory variable can only be accessed by the owner. But, there is an exception. Processes with data paths connected to another process data port can access that same data port. In effect the two data ports are linked together and their values will always be the same. The data ports can be though of as pointers pointing to the same memory location.

Data paths have direction. The significance of the direction of these data paths represented the read and write permissions of the processes to the ports. For example, in Figure 4.2, the Packet_Source process can write to the Transmitter process data port but it cannot read it. This means that whatever process Transmitter writes to it's data port, process Packet_Source can read it. To make Packet_Source be able to access the other data port, the data path should be bi-directional.

PROCESS DESIGN LEVEL

A collection of cooperating processes constitute the overall behavior of stations or nodes. Each process can be viewed as an *interrupt handler*, responding to certain events and doing some processing.

The processes represent the protocol of the network. Protocols are the set of rules followed by the network in its transmission and reception of data. As was previously mentioned GNM allows for modeling of protocols. Among the OSI Protocol layers, GNM concentrates at the lower layers, namely the *Network, Data link*, and *Physical* layers. The other layers can still be modeled but it would take a bit of effort.

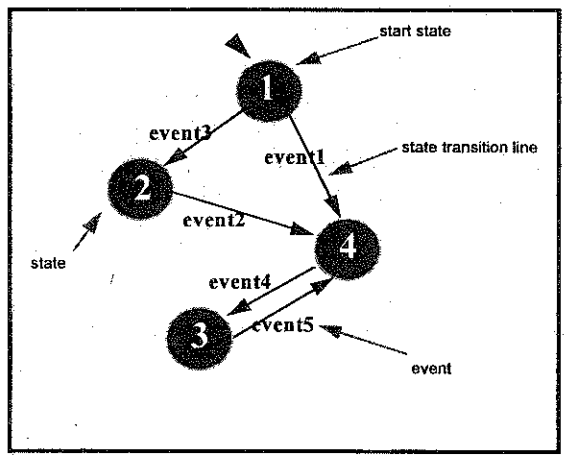


Figure 5. Process design example

In GNM processes are modeled with the use of **Finite State Machines** (FSM). Since FSM's are the de facto standard in modeling protocols, it is simply logical for us to adopt FSM's as the modeling paradigm. And because of the power of FSM's, GNM can support almost any type of protocol, resource, application, algorithm, or queuing policy.

A process is awakened, run and eventually terminated. A good understanding of the mechanisms by which a process keeps itself alive is essential for the development of effective network models.

Each process operates in a cycle consisting of the following steps:

1. a process is awakened by an event.
2. the process responds to that event and performs some specified operation.
3. it then puts itself back to sleep and waits for another event.

Before a process puts itself to sleep, the process first specifies the events that will wake it up again. A process that goes in to a state with no outgoing state transition has effectively terminated itself. Since it cannot specify any events to wake it up, the process will never wake up.

EVENTS

Events are the life blood of processes. Events wake up processes that constitute the overall operation of the node. In GNM events can be thought of as a two-tuple <who, event>, where who is the process that triggered the event and event is the actual event. This way processes can specify to whom it will respond to.

The who in the tuple can be unspecified. This will mean that the waiting process will be triggered by event no matter which process triggered it.

Events are triggered by processes. Any process can trigger any event it wishes, but the events triggered are only visible to processes within the same node. Processes cannot trigger processes from another station. For example, in the simple sender - receiver topology, processes at the sender node can not trigger an event for processes in the receiver node.

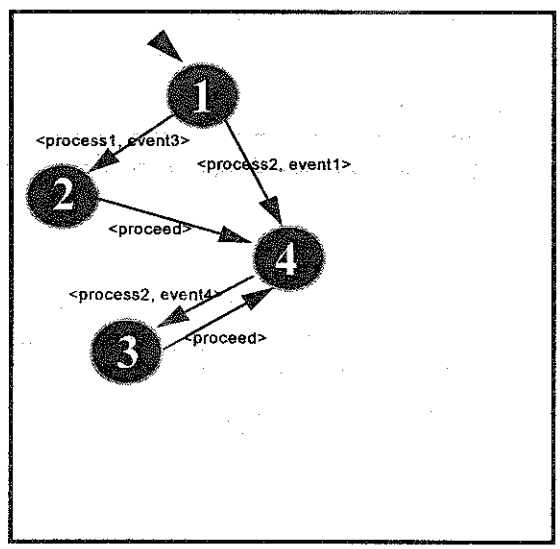


Figure 5.1. FSM and its events

Events don't have specified targets. All processes in the same node can see the event. It is up to the process whether it will respond to the event or not.

Process can trigger itself.

This way processes can have control in its own processing. Triggered events happen instantaneously. This means that if a process triggers an event a time 1, all the other processes

will see the event at time 1. Listed below are the processes that trigger events.

- the time process
- the port processes
- any other process

THE TIMER PROCESS

All processes in GNM recognize a single indivisible unit of time. Time is discrete. In GNM, time is an integer number and time starts with 0.

Time is advanced by a global process called the Timer. The events triggered by the timer is seen by all processes regardless of which node it is associated with.

Events triggered by the Timer is of the form <Timer, n> where n is an integer number. This means that a process waiting for this event will be triggered after n time units from the time it is put to sleep. In effect, an event of this form will act like a simple time delay event.

THE PORT PROCESS

Another important processes is the port process. The port serves as the interface of the to the link which is in turn connected to other stations. It is through this port that the transmission and reception of data takes place.

The port process has the following parameters:

1. error probability (ep) - this is the probability that a received packet is damaged. The probability functions follows a normal distribution.
2. transmission rate (tr) - the transmission rate is rate at which a single bit is pumped through the port. For example, if the transmission rate is 2, then bits of data are transmitted every 2 time units.

The port process has a single data port. Upon receipt of a new packet the data port contains the data structure sent through the link. On transmission the data port does not have any use.

When a process wishes to transmit data, the process must specify which port to transmit it to. Also the process must specify the size of the data to be transmitted (in bits). This way the ports will be triggered when to transmit and can compute the length of transmission. The amount of time in transmitting the data given a specified length is given by the formula.

transmission time = pd + ((n-1) * tr)

where:
pd - propagation delay of link
tr - transmission rate
n - length of data/packet in bits

BLACK DESIGN LEVEL

GNM uses a hierarchical data flow block diagram as the graphical programming language for defining the states of the state machines. The language is adopted from the **BONeS Designer**. GNM adopted this language because its ease of use. Since each block performs a specific task, the developer is abstracted from the "how's" of the blocks. This way code can be easily reused. The processing of each block is also abstracted.

The block in GNM performs specific tasks. There are blocks that do mathematical operations. Some provide queuing, counting and even control flow. The number of tasks that a block can do are endless. The developer simply has to define this blocks for use.

The modules in the block diagram operate on data structures which propagate through the block diagram on connections between ports. These data structures can be of any of the primitive types or any user defined structures.

A module provides interfaces to other modules through inputs, outputs, and arguments. The figure below is an example of a module. A module can be specified in one of two ways: either by primitive code or by an internal graphical structures.

The interface of a block to other modules are through its ports (ports here are different from the ports of the station discussed before). Input ports provide a way for data structures to enter the module while output ports provide a way for the module to pass data structures to other modules.

The execution of the block diagram is fairly simple. It can be compared to a Petri Net diagram. Every time a block receives a data structure from all of its input ports, the block is executed. The block will only execute if all of its input ports are enabled.

Then after execution the block sends its output through its corresponding output port where it triggers another block. This process continues until no more block is enabled.

In GNM states are treated like blocks, except that it does not have any input or output ports like those shown in Figure 6. It also have two, blocks stop and start. These blocks serve as terminators of the diagram.

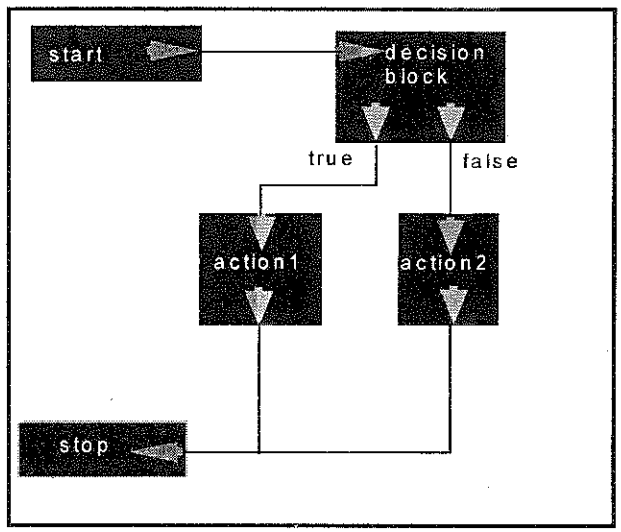


Figure 6. Block Diagram

When the state is executed, the start block automatically triggers. In effect, the whole diagram goes into execution. The stop block on the other hand functions like a sink. Data that goes in this block does not go out again.

At the lowest level of the design hierarchy are the block primitives. These are blocks that are already implemented as code. GNM does not specify what type of language to use at the lowest level. Depending on the implementation, the block primitives can be written in any language.