*Proceedings of the 1995 Winter Simulation Conference*
*ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman*

# MODULAR MODELING FOR NETWORK SIMULATION LANGUAGES: CONCEPTS AND EXAMPLES

Charles R. Standridge

Department of Industrial Engineering
Florida A & M University - Florida State University College of Engineering
2525 Pottsdamer Street
Tallahassee, FL 32310, U.S.A.

## ABSTRACT

Modular modeling supports the fundamental engineering strategy of dividing problems into component parts, solving each part, and linking the solutions together. We discuss the concepts and capabilities necessary to build modular network simulation languages and apply them to problem solving. Necessary modeling constructs are defined as a part of a new modular network simulation language, ModNet, and examples are given.

## 1 INTRODUCTION

Dividing problems into parts, solving each part, and linking the solutions together is a fundamental engineering tactic. This procedure has not been supported by the network modeling approach employed by some well-established simulation languages such as SLAM II (Pritsker 1987; Pritsker, Sigal and Hammesfahr 1994). Supporting such a problem solving strategy in the context of a network simulation language seems like an important step forward.

Modular modeling is one proven method supporting the above approach to problem solving. A model is comprised of individual components called modules. As expressed by Cota and Sargent (1992), a model module should have two properties, locality and encapsulation. Locality means that all related operations, including decisions, are found in one place. In modeling terms, this implies that all interactions between particular entities are isolated in one module. Encapsulation means that the model of these interactions can be changed without impacting models of other entity interactions.

The benefits of modular network modeling include the following:

1. A more structured modeling framework rather than a single network which is often large, complex, and confusing.
2. Support for model development by a team whose members work in parallel. Each member of the team can build modules independently as long the communication mechanisms between the modules are well defined.
3. Support for the development of libraries of modules. Simulation software developers can supply network modules for direct use or adaptation. Simulation application experts can build up libraries of modules to shorten the development cycle for new models.
4. Increased ease of learning for new simulation users who can learn smaller, well defined modules instead of complex (at least in the perspective of a new user) networks.

This paper describes the concepts and modeling constructs of a modular network simulation language. Examples illustrate the concepts. First a brief review of the modular modeling literature is given.

## 2 LITERATURE

Various approaches to modular modeling have been attempted over a number of years. These approaches include the use of macros to allow a model of a system component to be replicated within traditional process languages, the application of libraries of models expressed as macros in graphical model building to help tailor modeling constructs to classes of systems, as well as the use of object-oriented and artificial intelligence techniques for building modular modeling languages.

Pritsker (1977) describes Q-GERT subnetworks. A modeler could define a portion of any network as a subnetwork which could be replicated anywhere within the network in which it was originally defined. The duplicated subnetwork could be edited by deleting existing constructs and inserting additional ones. An analogous macro capability is provided by GPSS (Schriber 1991).

Gordon, et al. (1991) describe a graphical simulation language in which user defined elements can be

constructed to customize the set of modeling constructs to particular contexts. These user defined elements are specified as macros built from the existing modeling constructs. The authors state that this capability supports hierarchical modeling and provides an extensible language. The commercial software package ARENA (Pegden and Davis 1992) provides similar capabilities.

The classic modern definition of modular modeling is given by Zeigler (1990). Atomic components correspond to modules. Atomic components may have a number of input ports. An action is taken whenever a message arrives on an input port. An atomic component may not directly access the state of another atomic component.

Cota and Sargent (1992) define a modification of the process interaction world view to better support modular modeling. The preemption of the activities of one process by another prevents proper encapsulation. The modification provides an alternative to this pre-emption that supports encapsulation.

The Hierarchical Simulation Language (Sanderson, et al. 1992) is a simulation language that supports process world view modules. The language structure resembles Pascal and belongs to the same category of simulation languages as SIMULA (Birtwhistle, et al. 1975). The language associates a particular class of transactions with each module, that is each module can process only one type of transaction which must be defined in other modules that invoke it.

Nadoli and Biegal (1993) present a knowledge representation scheme to achieve modular modeling. Blackboards are used to model how intelligent agents make complex decisions in the operation of a manufacturing system. The manufacturing system is represented by classes of queueing networks.

Joines, Powell, and Roberts (1992, 1993) and Joines and Roberts (1994) described a simulation engine constructed using the object-oriented paradigm and implemented in C++. The implementation of a network simulation language, YANSL, based in this engine is described. The engine is modular and extensible. For example, the engine could be extended to provide a new type of branching from network nodes by defining a new branching class. Additional nodes could be defined using existing classes for transaction arrivals to nodes and branching from nodes as well as new classes for node specific processing.

## 3 CONCEPTS

Modular modeling has to do with dividing a model into component parts called modules. Communication between modules is performed in a well defined manor that minimizes the knowledge modules must have about each other. Each module should represent a single well-bound interaction between a small number of physical or logical entities in the system under study such as the operations on parts performed at a work station.

The concepts that were used to guide the design of a modular network simulation language are as follows:

1. The primary types of entities in a network model are transactions and resources. A network module models the interaction of one transaction type with any number of resources.
2. Each transaction type is contained within one network module and there is only one transaction type per module. Transaction attribute values may be communicated between modules.

This prevision eliminates cloning of transactions, that is copying a transaction and processing the copy and the original simultaneously in the same network. The processing of the original and the cloned transactions should be modeled in separate modules.

3. Each resource and its possible states are defined within one network module. The name of the resource may be passed to other modules where state transitions are specified. This allows, for example, the same worker to perform tasks at multiple work stations where each work station is modeled by a different module.
4. All simulation outputs are defined in a single "main" module.
5. A modular model is not flattened for implementation purposes. Thus, the simulation engine always knows what module is being simulated which facilitates error reporting and observation of performance measures.
6. The object-oriented paradigm is the guide for modularizing a network model.

Figure 1 shows the structure for a model module based on these six concepts. An instantiation of a class corresponds to a module. A new class may be built from a pre-existing class. There are eight predefined methods for each class. All of the methods are private except for NETWORK and MEASURES which are public. The DATA COLLECTION and EXPERIMENT methods are specified only for one module per model which acts as the "main" module. All methods are optional except for NETWORK. The use of the methods will be illustrated in the following sections.

## 4 CONSTRUCTS

The Modular Network Language (ModNet) implements, demonstrates, and illustrates the modular network concepts defined in the previous section. The

```
name CLASS BASE class_name
INITIAL CONDITIONS
  initial state variable values
DATA COLLECTION
  variable names
EXPERIMENT
  run length, number of replicates, etc.
VARIABLES
  type name
MEASURES
  variable = expression
TRANSACTION
  data structure defining attributes
RESOURCES
  definition including state transition rules
NETWORK parameter list
  network
END CLASS DEFINITION
```

Figure 1: Modular Network Class Paradigm

particular ModNet modeling constructs that embody these concepts as well as those employed by the examples in the following section will be defined. A complete definition of the language will not be attempted here.

### 4.1 Entity Definition

A ModNet network describes how a transaction is processed by a system where resources are scarce and can constrain processing. For example in a manufacturing system, a transaction can represent a part and resources the machines and labor required to perform operations on that part.

In ModNet, a transaction is defined by a structure of numeric and character attributes whose values distinguish it from other transactions. Consider the following example:

**Transaction:**

| | |
|---|---|
| **NUMERIC** | **time_of_arrival** |
| **CHARACTER** | **part_type** |
| **NUMERIC** | **position_on_route** |

The transaction processed by a particular module represents a part and has three attributes that tell when in simulated time the part arrived for processing, the type of the part, and at what station the part currently is located on its route through the system.

A resource may be in any one of several modeler defined states. (ModNet has no standard resource states such as BUSY and IDLE.) For example, a resource could represent a machine that is either BUSY, IDLE, or BROKEN. A resource could represent multiple identical and, for modeling purposes, indistinguishable machines. The number of units of the resource would be equal to the number of machines. A resource representing four such machines could be defined as follows:

| **Resource:** | **machine** | |
|---|---|---|
| | **INITIAL NUMBER** | **TRANSITION** |
| **STATES** | **OF UNITS** | **RULE** |
| **broken** | **0** | |
| **busy** | **0** | |
| **idle** | **4** | **PRIORITY (Q-BROKEN, Q-BUSY)** |

A resource named machine is defined with three states (BUSY, IDLE, BROKEN) and with four units all of which are initially in the IDLE state.

A resource unit is content to remain in some states, called active states, and tries to immediately leave other states called passive states. A transition rule for leaving passive states must be provided. In the example, BUSY and BROKEN are active states. A resource unit will remain in either of these states until the model changes it to another state. On the other hand, IDLE is a passive state. The transition rule specifies that upon entering the IDLE state a resource unit will immediately seek a transaction to process, first one that is waiting in Q-BROKEN and next in Q-BUSY if there is no transaction waiting in Q-BROKEN.

### 4.2 Module Definition and Reference

The first node in any ModNet network, the DEFINITION node, gives the name of the class to which the network method belongs, the base class from which the new class is derived, and the network parameters as shown in Figure 2. Since there is only one network per class and it is always clear from the context that the network method is being referenced, the class name serves as a unique identifier of the network.

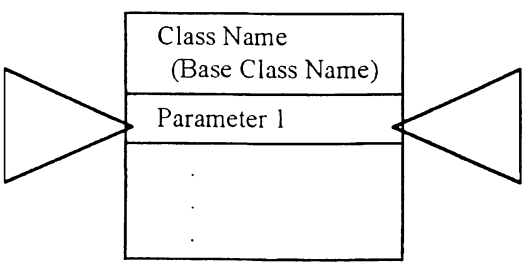


Figure 2: DEFINITION Node

A network is referenced and a new object instantiation of the class to which the network belongs is created if necessary by use of a REFERENCE node as shown in Figure 3. Parameters of the reference node are the class name containing the network, values for the network parameters, and the name of the instantiated object. The transaction reaching the REFERENCE node may wait until processing in the referenced module is completed or continue in the network in parallel with processing in the referenced module (/p following the class name).

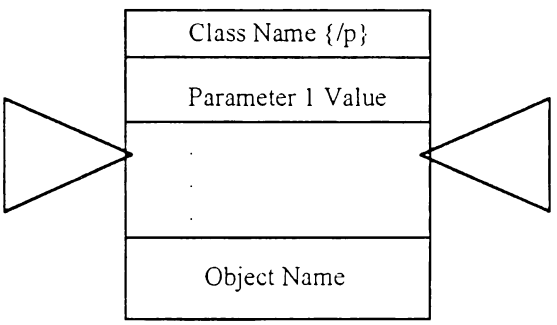


Figure 3: REFERENCE Node

### 4.3 Communication Between Modules

The DEFINITION and REFERENCE nodes provide the mechanism for supplying parameter values to a network. The two mechanisms for making information available from a network to all other networks are known as MEASURES and SIGNALs.

A MEASURES method gives a name by which a performance measure computed within a network may be referenced by all other classes. This is analogous to a method in object-oriented programming whose sole purpose is to make a variable value available to other classes without allowing them to change the variable value. For example, the MEASURES method:

**MEASURES**
**NINQ1 = NQ(WS1)**
**END MEASURES**

allows the number of transactions in the queue with name WS1 to be reference by other modules as NINQ1.

A SIGNAL node "broadcasts a message" that something of importance has just occurred in a network (an event) to which other networks may want to respond. Values may be associated with a SIGNAL and referenced in any network receiving the SIGNAL. A SIGNAL is referenced by the ModNet variable SIGNAL(name) as a time delay on an activity. That is, processing of a transaction ends when the particular SIGNAL is broadcast. The form of the SIGNAL node is shown in Figure 4.

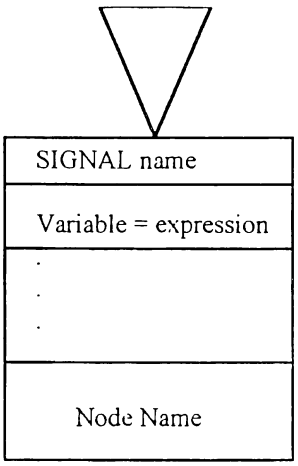


Figure 4: SIGNAL Node

### 4.4 Acquiring a Resource and Changing Its State

Any number of the units of any number of resources can all be changed from one state to another using a CHANGE STATE (CS) node. If a transaction may need to wait until units of a resource enter the desired state, an associated QUEUE node must be provided. Note that while most network simulation languages provide two node types for dealing with resources, such as the AWAIT node for acquiring a resource and a FREE node for giving up the resource in SLAM II, ModNet requires only the CS node for all such operations on resources.

The form of a CS node is shown in Figure 5. Each resource whose units are to change state is listed along with the state to change from, the state to change to and the number of units. In addition, the allocation rule governs the case when not all units of all resources are in the specified from states at once. The modeler may write an allocation rule in the form of IF-THEN-ELSE statements. Alternatively, a standard rule may be used. The standard rules are:

**JOINT** - Change the state of the units of the resources only when all units of all resources are in the specified from states.

**GREEDY** - Change the state of the units of any resource whenever the units are in the from state.

**SEQUENTIAL** - Change the state of the units of the resources in the order listed from top to bottom.

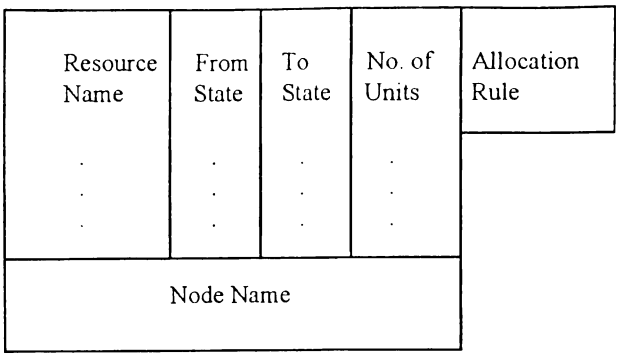


Figure 5: CHANGE STATE (CS) Node

### 4.5 Miscellaneous Constructs

The examples in the next section also use the following ModNet constructs that are common to most network simulation languages.

1. A CREATE node with parameters time between arrivals and marking attribute name. Specifying the marking attribute name causes the assignment:

**marking attribute = current simulation time.**

2. An ACTIVITY with two parameters: time delay and condition for performing the activity.
3. An END node that terminates a transaction.
4. A QUEUE node with parameter transaction ordering rule where transactions awaiting resources reside.

## 5 EXAMPLES

The following examples illustrate the basic modular network modeling concepts and how these concepts are implemented in ModNet.

### 5.1 Single Work Station

A typical work station consists of an input buffer where inbound parts wait for processing, a work area where inbound parts are transformed and an output buffer where outbound parts wait for transportation to the next work station. A model of a such a work station could consist of three modules: one for the arrival process named ARR, another for work station operations named WS1, and a third Main module to reference the other two.

The following concepts are illustrated in this example.

1. A SIGNAL node is used in the ARR module to indicate that a part has reached the work station. The signal is received in the Main module. Thus, the arrival process can be changed without changing the Main or WS1 modules.
2. The MEASURES method returns performance measure values of interest to the Main module from the WS1 module.
3. A REFERENCE node passes parameter values from the Main module to the ARR and WS1 modules.

The Arrive class from which the ARR is instantiated consists only of the NETWORK method shown in Figure 6. The parameter of the network is AvgTBA, the average time between transaction arrivals. A CREATE node models the generation of an arrival followed by a SIGNAL node which tells that the arrival has occurred.

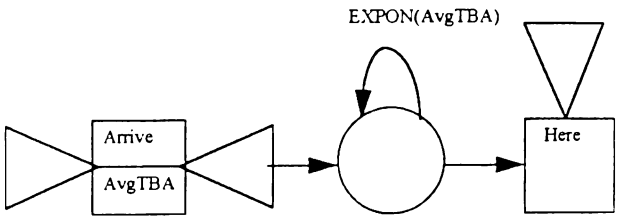


Figure 6: Arrive Class for Work Station Example

The STATION class from which the WS1 module is instantiated, Figure 7, has the NETWORK, TRANSACTION, RESOURCE, and MEASURES methods. The parameter of the network is OpTime, the operation time for a part transaction at the station. OpTime is a transaction attribute. The resource named WS has two states, BUSY and IDLE. Initially there is one unit of WS in the IDLE state. Performance measures are the number of transactions queued for the work station, NINQ, and the utilization of the work station resource, UTIL.

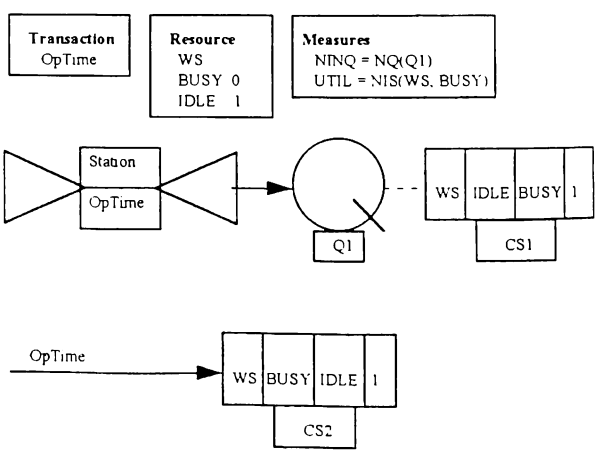


Figure 7: Station Class for Work Station Example

In the network, a transaction waits in QUEUE node Q1 for the work station resource WS to become IDLE at CS node CS1. After the operation is performed, the state of the work station resource is made IDLE at CS node CS2.

The Main class has the NETWORK and DATA COLLECTION methods, Figure 8. The network consists of a REFERENCE node for the Arrive class that instantiates the module named ARR as well as a reference node for the STATION class that instantiates the work station module named WS1. The SIGNAL name HERE from the module WS1 is referred to as ARR.HERE. Thus whenever a transaction arrives in ARR, a transaction is generated at the CREATE node in the MAIN class.

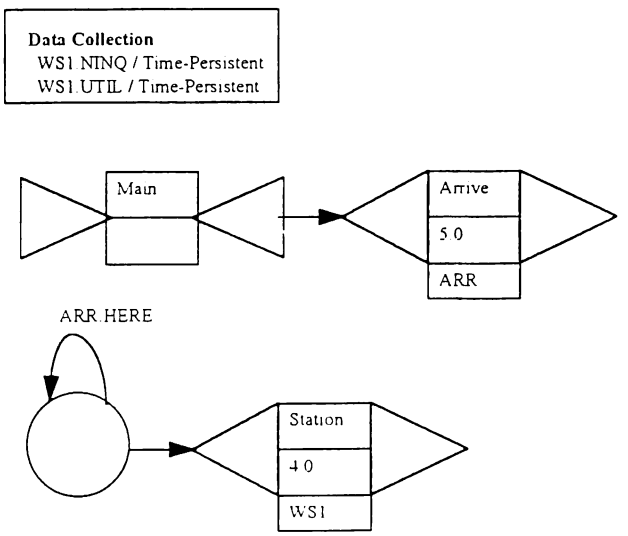


Figure 8: Main Class for Work Station Example

The DATA COLLECTION method collects the values of the variables in the MEASURES method of the WS1 module: WS1.NINQ and WS1.UTIL.

### 5.2 Serial System

A serial manufacturing system consists of a sequence of work stations with parts moving between them via some material handling device. In an unpaced system, the expected operation times for parts vary between work stations. Thus when a work station completes its operation on part, the following work station may still be processing a part. By placing the completed part in a buffer between the stations, the preceding work station may begin processing another part. If the buffer is full, the preceding station has no where to place the completed part and cannot begin working on another part. In this case, the work station is in the BLOCKED state. Time in the BLOCKED state is unproductive and may lessen system throughput, the number of parts completed per unit time. Preventing blocking requires buffer space that may be scarce. Thus, BLOCKED time and buffer space trade off against each other.

In this example, a four station serial system is modeled. There is a buffer of finite size between each pair of stations. Sufficient storage space exists preceding the first station. One module models the entire line. It references four modules, one for each work station. The first and last station are special cases. The first station has no preceding buffer and the last station no following buffer. The three classes of work station modules are First_Station, Last_Station, and Inter_Station. The part arrival process is modeled in a separate module, ARR. A Main module organizes the others.

The following concepts are illustrated in addition to those used in the first example.

1. The model of the four station serial system uses variations of the work station class Station developed in the previous example.
2. The special case station classes, First_Station and Last_Station, are derived from the more general class, Inter_Station.
3. The states of units of multiple resources are changed concurrently at CS nodes.
4. Resources common to a multiple work stations are defined in the module modeling the entire line. The resource name is passed to the work station module from the defining module.
5. The same Arrive class developed in the first example is used in this example.

The Inter_Station module is shown in Figure 9. The network has three parameters: the operation time at the station, OpTime; the name of the resource modeling the preceding buffer, BUF_LAST; and the name of the resource modeling the following buffer, BUF_NEXT. The only attribute of a transaction is OpTime. In addition to the two resources modeling buffers, the resource WS represents the station which can be BUSY working on a part, IDLE, or BLOCKED waiting for a completed part to move to the following buffer.

A transaction representing a part waits in QUEUE node Q1 for the work station resource to become IDLE. As modeled in CS node CS1, the part can then leave the inter-station buffer and begin processing on the work station. Thus, one unit of the work station resource becomes BUSY and one unit of the inter-station buffer becomes IDLE. The ALLOCATION rule in node CS1 is SEQUENTIAL to enforce the requirement that the part needs the IDLE work station before giving up its space in the buffer. The part is then processed by the work station. Upon completion of processing, the part

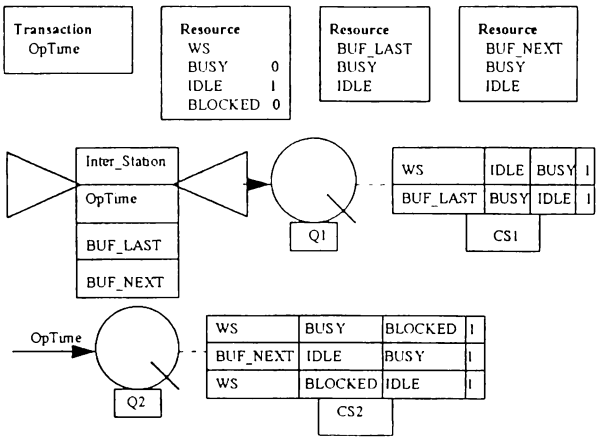


Figure 9: Inter_Station Class of the Serial Line Example

transaction enters QUEUE node Q2 associated with CS node CS2. In sequence, the work station resource enters the BLOCKED state and the part transaction awaits an IDLE unit of the resource BUF_NEXT representing an open space in the following buffer. When this space becomes available, the transaction leaves the work station and the work station resource enters the IDLE state.

The First_Station class, shown in Figure 10, is derived from the Inter_Station class. The only changes are in the network. The BUF_LAST parameter is not needed since there is no buffer preceding the first station. The state change concerning this resource is removed from the CS node name CS1. In a similar fashion, the Last_Station class is derived from the Inter_Station class. The BUF_NEXT parameter is removed as is the state change concerning this resource at node CS2. All other methods are the same as in Inter_Station.

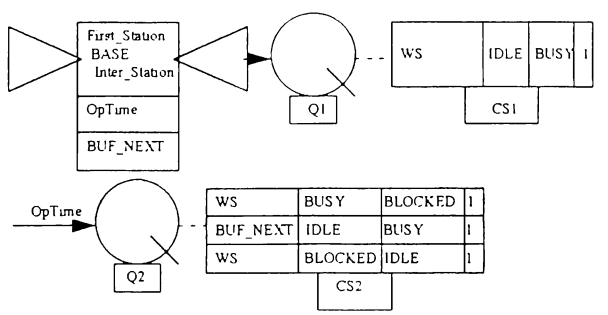


Figure 10: First_Station Class of the Serial Line Example

The LINE class shown in Figure 11 has three RESOURCES that model the three buffers. The MEASURES method makes the number of busy units of each buffer resource available to other modules. The NETWORK consists of one DEFINITION node and four REFERENCE nodes, one for each work station. The First_Station and Last_Station classes are instantiated once and the Inter_Station class twice to model the second and third stations.

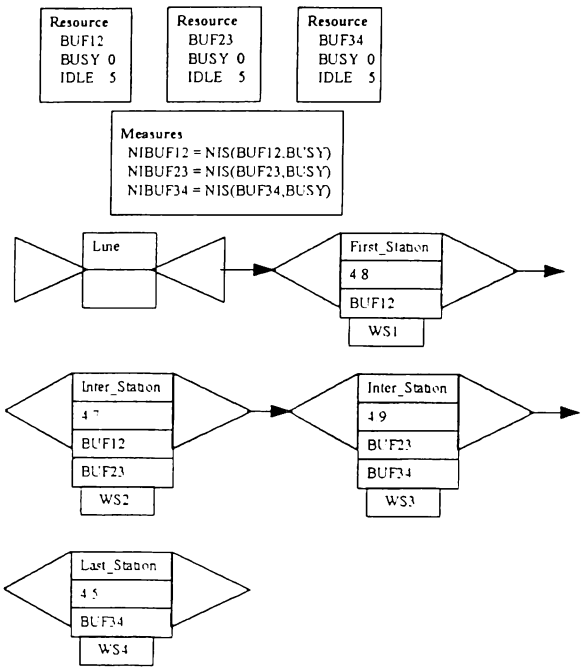


Figure 11: Line Class of the Serial Line Example

The Main module shown in Figure 12 is similar to the Main module for the previous example. An object with the name ARR is instantiated from the Arrive class to model the arrival process to the system. A CREATE node generates a transaction with attribute TA given the value of the current simulation time whenever ARR signals that an arrival has occurred. Next, the Line class is employed to simulate the operations of the serial line. Finally, an END node models the departure of the transaction. The Main module gathers statistics on the number in each buffer and the time transactions spend in the system.

## 6 SUMMARY

One way of bringing the benefits of modular modeling to network simulation languages has been discussed. Concepts for modular network models have been presented. The object oriented paradigm provides a guide to the design of a module consisting of standard methods for defining entities, data collection,

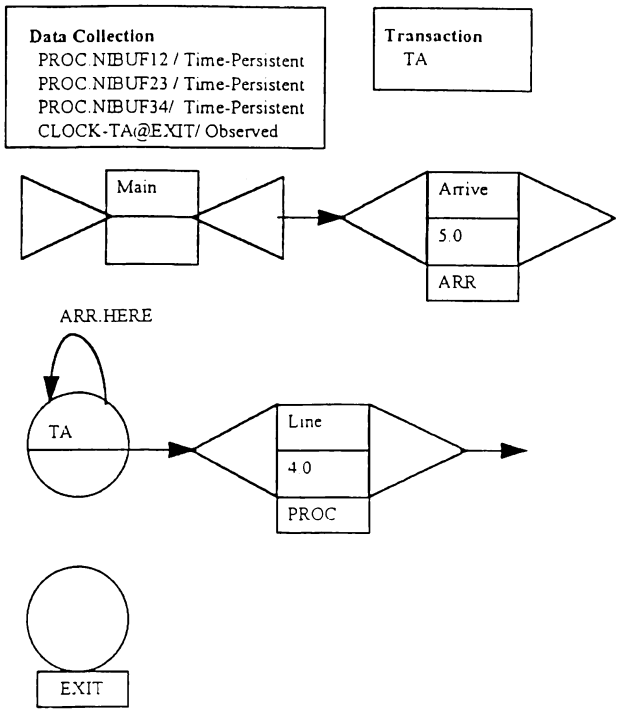


Figure 12: Main Class for the Serial Line Example

communication between modules and experiments as well as a network model. ModNet is a new modular network simulation language that is based on these concepts. Modeling constructs particularly needed for modularity have been defined and example models are given.

## REFERENCES


Birtwhistle, G. M., O. J. Dahl, B. Myhrhaug, and K. Nygarrd. 1975. *SIMULA Begin*, New York: Petrocelli/Charter.

Cota, B. A. and R. G. Sargent. 1992. A modification of the process interaction world view. *ACM Transactions on Modeling and Computer Simulation* 2: 109-129.

Gordon, K. J., R. F. Gordon, J. F. Kurose, and E. A. MacNair. 1991. An extensible visual environment for construction and analysis of hierarchically-structured models of resource contention systems. *Management Science* 37: 714-732.

Joines, J. A., K. A. Powell, Jr., and S. D. Roberts. 1992. Object-oriented modeling and simulation with C++. In *Proceedings of the 1992 Winter Simulation Conference*, ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, 145-153. Institute of Electrical and Electronic Engineers, San Francisco, California.

Joines, J. A., K. A. Powell, Jr., and S. D. Roberts. 1993. Building object-oriented simulations with C++. In *Proceedings of the 1993 Winter Simulation Conference*, ed. G. W. Evans, M. Mollaghasemi, E. C. Russell, and W. E. Biles, 79-88. Institute of Electrical and Electronic Engineers, San Francisco, California.

Joines, J. A. and S. D. Roberts. 1994. Design of object-oriented simulations with C++. In *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, 157-165. Institute of Electrical and Electronic Engineers, San Francisco, California.

Nadoli, G. and J. E. Biegel. 1993. Intelligent manufacturing-simulation agents tool (IMSAT). *ACM Transactions on Modeling and Computer Simulation* 3: 42-65.

Pegden, C. D. and D. A. Davis. 1992. ARENA: a SIMAN/CINEMA-based hierarchical modeling system. In *Proceedings of the 1992 Winter Simulation Conference*, ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, 390-399. Institute of Electrical and Electronic Engineers, San Francisco, California.

Pritsker, A. A. B. 1977. *Modeling and analysis using Q-GERT networks*. New York: Halsted Press.

Pritsker, A. A. B. 1987. *Introduction to simulation and SLAM II*. 3rd ed. New York: Halsted Press.

Pritsker, A. A. B. C. E. Sigal, and R. D. J. Hammesfahr. 1994. *SLAM II network models for decision support*. Palo Alto: The Scientific Press.

Sanderson, D. P., R. Sharma, R. Rozin, and S. Treu. 1991. The hierarchical simulation language HSL: a versatile tool for process-oriented simulation. *ACM Transactions on Modeling and Computer Simulation* 1: 113-153.

Schriber, T. J. 1991. *An introduction to simulation using GPSS/H*. New York: John Wiley and Sons.

Ziegler, B. P. 1990. *Object oriented simulation with hierarchical modular models*. New York: Academic Press.


## AUTHOR BIOGRAPHY

**CHARLES R. STANDRIDGE** is an associate professor in the Department of Industrial Engineering at the Florida A&M University - Florida State University College of Engineering. He led the development of the Simulation Data Language (SDL) and of The Extended Simulation Support System (TESS) for Pritsker Corporation. His current research interests are modular simulation environments, the integration of simulation into manufacturing design processes, and the analysis of health care delivery systems.