An Object-Oriented Model for Network Management

Soumitra Sengupta, Alexander Dupuy,
Jed Schwartz, Yechiam Yemini

CUCS-414-88

An Object-Oriented Model for Network Management

Soumitra Sengupta, Alexander Dupuy, Jed Schwartz, Yechiam Yemini

Department of Computer Science
Columbia University, New York, NY-10027

December 1988

Abstract

Most networks today lack management tools that support automated identification of faults and bottlenecks and support effective recovery procedures. The NetMATE project, discussed in this paper, addresses the issues related to distributed network management of large, heterogeneous networks. NetMATE employs a modular, object-oriented approach to develop extensible management tools and a model for network information. The first prototype of the system confirms the elegance of the design.

1 Introduction

Networks have become a principal means for information sharing and transfer in large corporations and institutions. The recent direction towards large enterprise-wide computing is a direct consequence of the interconnection and the ease of availability of computing resources in large numbers. The complexity of interconnections has risen sharply in the past few years. Consequently, management of the connectivity subsystems has become a matter of paramount importance in order to support resilient and efficient information flow in today's networks.

A network, in simplistic terms, consists of physical connections between many computing entities, and sets of protocols for information exchange over these connections.

Research supported in part by DARPA contract #F-29601-87-C-0074

The number of computing entities, the richness of interconnections, and the heterogeneity of protocols, all contribute to the complexity of a network and its management. While geographic distribution and functional isolation of different domains within a network make it impractical to employ a centralized network management strategy, a distributed solution, by its very nature, adds to the complexity of monitoring and coordination of network entities.

Current solutions for network management are mostly restricted to a single set of protocols executing on a single platform of hardware and connections supported by a single vendor. Most networks, however, consist of a heterogeneous mixture of components that communicate using a variety of protocols. Even when it is possible to reconcile the heterogeneity, the volume of management information from the network itself can be overwhelming; which makes the task of extracting correct and meaningful conclusions extremely hard.

The NetMATE (Network Management, Analysis, and Testing Environment) project at Columbia University addresses these and other issues related to effective network management. The fundamental purpose of NetMATE is to provide a model of a network, whether real or simulated, that can be manipulated and studied. NetMATE is being developed as a set of software tools that monitor the functional and performance behavior of large scale distributed network systems, analyze these behaviors, identify and isolate erroneous operational conditions, report them in a meaningful manner to operators, and support operator-directed, semi-automated network management. NetMATE is a natural extension of NeST, Columbia's Network Simulation and Prototype Tool system [1, 2], which is a portable simulation environment consisting of two parts: a monitor and a simulator. The monitor permits graphical iconic interaction and management of simulated network scenarios; and the simulator executes detailed simulated processes in accordance with the parameters set by the monitor and process definitions. NeST has been ported to over many sites including universities, industrial and governmental research and development laboratories, corporation development and operations teams.

In this paper, we discuss the design of a specific module in NetMATE which deals with the issues of modeling a network to create and store a permanent repository of generic and specific properties of network entities. Vbase, an object-oriented database [7], is chosen for the purpose of defining and storing the models of real network entities, their interrelationships, and dependencies. Vbase is an excellent and complete object-oriented software system with a clean architecture and supports a rich set of data value and aggregate types (and associated operations) and relationships. One of the most useful features of Vbase is that the types themselves are objects, and hence can be manipulated to a very high degree. Given the intricate structure and behavior of a network, Vbase has been a very useful tool and database in implementing the NetMATE network model.

The remainder of the paper is organized as follows. In Section 2, we specify the functional requirements of NetMATE and discuss the sophistication required to fulfill the requirements. NetMATE architecture is presented in Section 3, where we discuss the different modules that support the required functionality. Section 4 elaborates on the data model. Specifically, the generic relationships between the network entities and their representation in the data model are detailed. We also briefly touch upon the representation of information required by various modules of NetMATE. A discussion of the Vbase database with respect to the NetMATE project follows in Section 5. The current status of NetMATE is presented in Section 6, and we conclude in Section 7.

2 Functional Requirements

The functional requirements of the NetMATE system are derived from the needs of various users of the system such as operators, administrators and researchers. There are few fundamental necessities, however, that are desired by all users. In order for the NetMATE to be a generic tool for network management, it must support these functions, and more importantly, be extensible in accommodating future needs.

Configuration Management — A network management system should be able to define, confirm, and exercise control over the configuration of network entities. Automatic validation of a configuration should be possible which includes validation of connectivity and other network relationships. For instance, the system must recognize that a standard *modem* cannot be attached to a generic *ethernet* but it can be connected to an *asynchronous line*; and if a *modem* is defined to be connected to an *asynchronous line*, then the system must confirm the actual connection in the network.

Configuration Management requires definition of static information regarding generic network entities such as modems, hosts, terminal servers, link protocols, transport protocols, and so on. Furthermore, it requires definition of properties of specific entities within a generic class to check the validity of a configuration accurately.

Fault and Performance Management — NetMATE functions should include fault management facilities such as fault tracing and reporting, event logging and execution of diagnostic tests. Similarly, it should be able to monitor and evaluate the performance of the network by collecting and disseminating performance data and maintaining performance logs. The provision of these functions require real-time management of large amounts of network information by NetMATE, and an expert system capable of diagnosing faults and performance degradations.

Flexibility and Extensibility — NetMATE must be flexible enough to accommodate the wide variety of vendor products, connection schemes and protocols abundant in today's heterogeneous networks. The generic nature of NetMATE tools must provide mechanisms to adapt to such diversity with ease, and provide both generic and specific reasoning about configurability and fault analysis. One of the important strategies of NetMATE implementation is to take advantage of existing network management systems that control only a single, homogeneous network (*e.g.*, **NetView** - an SNA network management system [8]).

Distributed Network Management — In a large network, it is impractical to enforce a centralized network management system for mainly two reasons: Vulnerability of the central system failure; and an overwhelming flow of network management information that would slow the real-time management and fault analysis processing. The distribution of management functions, however, gives rise to typical distributed systems' problems such as concurrent network commands (updates) issued to the same entity by two different management systems. NetMATE systems should have clear and possibly non-overlapping management domains, and a coordination policy to avoid such problems.

Research and Planning — Besides managing a real network, NetMATE should support design and evaluation of new networks and protocols through simulation of the same under various test conditions such as traffic patterns, loads, and failures. For real networks, this facility would allow a configuration to be tested before actually installing the network, and thus aid in the planning process. For researchers, the simulation techniques would be beneficial in modeling and verification of correctness and inter-operability of protocols.

The OSI model of network management [5] includes the first two functions as requirements. Two other functions are mentioned - Accounting and Security Managements. The NetMATE system, however, concentrates on the more fundamental requirements mentioned above, and we believe that the additional functions can be accommodated in NetMATE later.

3 Architecture

Let us examine the architecture of the NetMATE system in terms of its system modules and their functions. The functional capabilities of NetMATE mentioned above are not implemented individually by the modules; instead, a subset of NetMATE modules cooperate to support a specific function. This is desirable in order to make use of existing capabilities and to avoid duplication of effort.

Conceptually, there are five major components of a NetMATE installation: the *Physical Network*, the *Simulated Network* (or Simulator), the *User Interface* (UI), the *Expert Analyser* (EA), and the *Modeler*. The Modeler is the heart of the installation in the sense that it holds the network model and other information in a database; all other components function as clients to the Modeler. Thus the Modeler serves as the hub of the system, and provides the essential glue by being a central repository of information. The Modeler may interact with the Physical Network through *Network Control Points* (NCP) which control a specific subset of the network. In a distributed implementation, the Modeler also provides services to other installations in the system and takes part in joint coordination of information flow with other Modelers. Figure 1 depicts the NetMATE architecture.

NetMATE architecture policy dictates that each module individually be as independent as possible. This allows for distributed execution of the modules. (The Expert Analysis module, possibly, is the only exception due to efficiency reasons.) The Modeler, in addition to the network information, also holds information for each client, and thus acts as a database server for its clients. It also serves as a conduit for information flow between its clients.

The definition of interfaces between the Modeler and other client modules are of considerable importance. They are defined by a set of information transfer units, and the invocation procedures to initiate the transfer. Ideally, a Simulator-Modeler interface should be similar to a NCP-Modeler interface so that there are minimal differences between a physical network and a simulated network. Different UI-Modeler interfaces may implement different views of the network to different users (for security as well as user-sophistication level reasons). A Modeler-Modeler interface could include only a specific view (and thus have a superset of a UI-Modeler interface). The architecture of NetMATE is such that an interface may be re-used for different clients, thus providing ease of use and flexibility in the system.

NetMATE implements a network management control and information flow which is shown in Figure 2. A real network element is governed by NCPs which have detailed knowledge about the entity. (A simulated network entity is governed by the Simulator.) The NCPs (and the Simulator) relay information to the Network Manager(s) (implemented by the model in the Modeler and the logic in the Expert Analyser module). In absence of an explicit NCP, the Network Manager functions as an NCP to the entity. The Manager ultimately provides the information to the users through different views implemented by the UI-Modeler interfaces.

It is important to note that hierarchy depicted in Figure 2 does not necessarily have a tree structure. Indeed, for information flow, it may be desirable to send critical information to more than one Network Manager. However, for the control flow, which demands strong consistency, it may be necessary to superimpose a strict and unique hierarchy structure by which one network element is controlled by a single Network Manager.

4 The Modeler: An Object-Oriented Database

In this section, we elaborate on the model chosen for the network and other information as implemented by the Modeler. A network (real and/or simulated) is subdivided into *Network Management Domains* consisting of physical devices and connections. Selection of a domain is based upon any reasonable criterion, such as geographic proximity, or similar family of communication protocols (TCP/IP compatible, SNA-compatible, etc.) For control purposes, the management functions associated with a network entity may not overlap different domains. The network model in the Modeler represents a network management domain. Multiple Modeler systems interchange network information and carry out management functions cooperatively to meet the goals of the NetMATE project. The management domain approach is motivated by [5], and has appeared in other network management designs [3, 4].

A Modeler consists of a *Network Model*, which describes the network entities in the domain being managed; a *Network Management Model*, which defines the network control points and managers of sub-domains or peer domains; and an *Organization Model*, which contains information about the clients of the Modeler. Associated with each model, there is a group of cooperative processes that maintain the integrity of, and provide services for, the model. The models are realized by an object-oriented database, and collectively form the basis for an integrated network management tool for a specific domain.

4.1 Network Model

A Network model is partitioned into two modules: *Network Configuration*, and *Network Definition*. Network Configuration consists of model instances that correspond to the actual network entities in the domain. Network Definition consists of pre-defined classes for these entities from which the instances in the Network Configuration partition are realized. Consider an example where there exists a class called **Modem** in the Definition, and there are three modems in the domain. Then there exist three instances of the Modem class in the Configuration representing the three modems.

Network Configuration

Network Configuration stores two kinds of information: generic network topology, and specific information about network entities. While the specific information is realized from the class hierarchy defined in Network Definition, the topology information requires a generic class hierarchy of its own in the Network Configuration. This hierarchy is partially depicted in Figure 3.

The generic network entities in a domain are nodes, links, and protocol layers. A node object in the model represents a node entity which may be a machine, protocol engine, process or application in the network. Similarly, a link object represents a link entity which is a channel through which node entities can communicate. Therefore, node and link objects model abstract notions of endpoints and connectivity among the endpoints in the domain. The classes **node** and **link** are subclasses of a **simple** class, and a relationship called *connection* exists between pairs of nodes and links.

Since the behavior of node and link entities are governed by a pre-defined, layered communication protocol, the Network model defines a **layer** class of objects. A layer object represents the properties of an installed protocol and specifies a set of node and link objects in the Configuration partition that follow the protocol. A node or a link is contained exclusively in a layer, that is, they cannot be shared among multiple layers. This is an important concept in NetMATE as we conceive the endpoints of communication to be more than just physical devices and the connectivity between endpoints to be more than just a physical connection. Indeed, the nodes and links in the *physical* layer do represent physical devices and connections; but in a higher level layer, they represent endpoints and connections specific to that layer. For example, in a *X.25* layer, there may be *X.25* nodes and *X.25 Virtual Circuit* links. The model will usually include a number of layers with a clear semantics of ascendancy of layers.

A second level of abstraction within a layer is achieved by a group object (an instance of the **group** class) which represents a collection of nodes, links, and other groups in the same layer. This aggregation gives rise to a simple hierarchy with parent-child relationship. A group, as such, has no pre-defined network semantics. Its properties and functions are determined by its definition and use. The class **group** is a subclass of a **compound** class, and defines a *grouping* relationship between itself and its children objects. The classes **simple** and **compound** are subclasses of a **network_element** class.

A relationship called *membership* is established between layer and network_element objects representing the fact that a layer consists of nodes, links and groups. Additionally, a *mapping* relation is defined between objects in different layers. The *mapping* relationship represents the correspondence of objects at different levels of abstraction (layers) in a protocol suite. Objects (simple or group) at a "higher" layer map to objects (simple and/or group) at "lower" layers. The mappings, thus, model the functional dependency of the well-being of the objects at one layer to the well-being of the objects at lower layers, and by definition, determine the ascendancy of the layer objects. A subclass of **group**, called **mapgroup**, is defined to realize the *mapping* relationship. An object at a higher layer maps to a set of mapgroups, which individually represent the lower layer they belong to and the group of objects involved in the mapping in that layer.

The relationships *connection* and *grouping* are many-to-many, whereas the *membership* relationship is one-to-many. The *mapping* relationship is one-to-many when considered between a higher layer object and mapgroups. However, an object (simple or group) in a mapgroup may also be a member in other mapgroups in the same layer, and if considered between objects in different layers, the *mapping* relationship is many-to-many. These relations are shown in Figure 5, and are implemented as properties of the classes in the hierarchy. The instances of all objects representing the generic network entities in a domain and explicit representation of the relationships among the objects conceptually construct the topology of the network.

For each generic node, link, group and layer object, NetMATE makes use of additional specific details about their properties other than the generic relationships mentioned above. A node object may represent, for example, a host computer, a gateway, or a modem. Similarly, a link might represent an Ethernet bus, a virtual circuit, or a TCP connection. A gateway node may send management information which differs significantly from the information sent by a bridge node, and usually, they will support different sets of management commands. It is therefore necessary to associate specific information about the capabilities of the generic network entities, and this is accomplished by *kind_of* relationship for layer, simple, and group objects. The objects at the other end of the *kind_of* relation are instantiated from the class hierarchy defined in the Network Definition Model, and are the place holders for the specific properties.

Network Definition

Network Definition consists of (relatively) static information about the properties of layers and other network entities. It aids in construction of a Network model in two major ways: Creation of objects for specific properties in Network Configuration; and integrity maintenance of the defined model. In a specific Network model, only a subset of the Definition model that represents the actual network entities in the domain may be chosen to instantiate the specific properties in the Configuration model.

The Definition partition provides property templates (by defining classes and associated attributes), and defines restrictions on relationships (by specifying allowable values for attributes). Additionally, the operations defined for the classes provide network management commands such as querying for status. Thus the partition functions as a dictionary of entity-specific definitions of network management and control information supported by each entity. This is a very flexible approach which allows the different classification schemes to be introduced in the Definition model in a semi-dynamic fashion. One example of classification could be a vendor-specific management scheme. Figure 4 shows a sample Network Definition class hierarchy for the nodes in the domain. Once a generic node object is defined in the Network Configuration, it may be assigned a specific node class in the Network Definition, upon which one (or many) property templates are instantiated from the chosen class which are then linked to the generic node through its *kind_of* relationship.

The partitioning of the Network model into two modules allows NetMATE to provide generic services at the user level, which are subsequently translated into different sets of specific procedures for different entities with minimal knowledge requirement from the user. This flexibility is extremely important in an heterogeneous network domain because it allows common network analysis functions to be specified at a high level of abstraction, and then continually refined at the lower levels.

4.2 Network Management Model

The Network Management model represents the management structure, and is also partitioned, similar to the Network model, into two modules: Definition and Configuration. The Management model defines, and has instances of, Network Control Points which represent other network management systems. The protocols by which the Modeler communicates with an NCP is captured by the operations defined for that **NCP** class.

In a distributed network management implementation, a Modeler may request information from another Modeler. In this scenario, the latter Modeler behaves as a Network Control Point of the former. Therefore, many properties of the **NCP** class are applicable for peer Modelers acting as servers. This architecture is easily extendible to form a management hierarchy of Modelers consisting of local, peer and parent managers. In the Management model, one is interested in a **manager** class, a subclass of **compound**, instances of which represent a set of manager objects with a *managing* relationship. Note that **NCP** thus becomes a subclass of **manager** class (Figure 3). For NetMATE managers, the class will have the NetMATE Network Management Protocol routines defined in its operations and will include appropriate interface definition. An interesting aspect of the management model is that by specifying the Network Management Protocols as a set of layer objects, and nodes and links that implement the management activities in the Network model, it should be possible to monitor the performance of the Network Management Protocol itself.

It is useful to differentiate between Network Management model information and the Network model information for various reasons. Management information is required for the distributed implementation of NetMATE, and also to model NCPs. The hierarchical structure reduces the data flow between every network entity and a single Modeler (and thus supports filtering), and allows the integration of vendor-specific network management subsystems in a uniform fashion.

4.3 Organization Model

The Organization model keeps information about the various client modules of NetMATE that are using the services of the Modeler. This model is also used by the clients, if desired, to store information related to their operations. Not all information in this model related to a client need be persistent; it may be sufficient to keep the information as long as the client requires services from the Modeler. While a detailed design of the Organization model has not been finalized, some of the important issues follow.

Important information related to a client include communication protocol and associated interface, outstanding requests, outstanding replies, and event triggers. Among these, the outstanding requests and replies imply asynchronous, message-based communication protocol between clients and the Modeler. Event triggers require a mechanism for definition of event objects (which may be persistent), evaluation of associated procedures and appropriate interruption of the client. Additionally, the Organization model routines are responsible for maintaining the consistency of the Network and Network Management models as they are accessed by the clients. A partial class hierarchy for the User Interface appears in Figure 3.

A user interface client may choose to store display information in the Organization model that corresponds to the objects in the Network model. The class hierarchy defined for display information includes **view** (instantiates a window object), **viewgroup** (instantiates a group containing objects that are displayed in a view), and **appearance** (instantiates an object representing the graphical image of an element on the screen). A set of processes in the Modeler may be dedicated for the user interface client for manipulation of such display objects.

The major function of modules in an Expert Analyser is to provide automatic decision making in fault and performance bottleneck analyses. For this, an Expert analyser must understand the Network model in great detail. It is the principal client of the event reporting services, and the principal requestor of information from the network.

An Expert Analyser consists of a generic knowledge base which consists of rules for problem determination in a generic model of a network such as the Network model described earlier. However, in order to be applicable on a live (real or simulated) network, many rules would require network entity-specific attribute values to be substituted in real time. In some cases, where generic rules are not possible, the rules must be entity-specific to effectively handle a entity-specific problem. Thus an Expert Analyser client may consist of rules that are best defined in the Network Definition, along with other entity-specific attributes and operations. Although these rules would be a part of the Network model information, their invocation and evaluation will be controlled by the Expert Analyser client.

5 The Modeler and Vbase

The preceding description of the Modeler architecture clearly demonstrates the need for an object-oriented system supporting an object-oriented data model for network management systems. A relational model for elements in a network is impractical and unwieldy mainly due to heterogeneity, and large number of, and different, ways network elements have to be managed. For example, the properties of, and applicable operations on, a node differs widely depending upon the layer it belongs to (and hence the protocol it executes), its own type hierarchy within a layer (for instance, whether it is a **gateway**, or a **bridge**, or a **host** object in the *physical layer*), and possibly the vendor network management facilities it provides. Since NetMATE is designed as a generic set of tools, it is therefore necessary to isolate these dependencies in different collections of properties and associated set of specific operations for ease and portability in network management. Vbase supports the required modularization capability of the NetMATE models to a very high degree and has been an invaluable tool for developing the NetMATE prototype.

Vbase, as a general-purpose object-oriented database, supports properties, operations, and inheritance for abstract data types defined in a type hierarchy. Besides using the basic features to define the network element types in NetMATE, we have made good use of the convenient *many-to-many* relationship construct in Vbase (implemented as "distributed set"). The exception and trigger concepts, rarely found in other systems, have proven quite valuable in the Modeler development. Although at the current prototyping stage we have had no need for concurrency control, the synchronization mechanisms available in Vbase also seem quite adequate for that task.

The clean structure of Vbase is most evident in its design that the same constructs that Vbase supports, are also used in its implementation. One useful consequence of this is that the user-defined types and methods themselves are *objects* of system types **Type** and **Method**. We have exploited this feature by being able to use the object id of a type at runtime to efficiently refer to, and exchange information about, that type across computers and disparate object-oriented systems.

Some desirable Modeler functions that are not currently possible through Vbase follow. Dynamic modification of the type hierarchy and definition in the Network Definition model, even in a restricted form, is required. We expect this function to be supported by the object-oriented database through dynamic schema updates. Secondly, it is desirable to have a Structured Query Language (SQL)-like features in the interface between different modules and the Modeler. This can be made possible by having the Object SQL queries invoked and evaluated from the procedures in Vbase. Finally, a distributed implementation of Vbase (which probably is more of a research issue) will be extremely helpful in the distributed NetMATE system.

6 Current Implementation

The NetMATE system currently has prototypes for a User Interface module and a Modeler. The User Interface module is implemented in the X Window system and is written in C++. It makes use of the Interviews window systems interface library. At present, dynamic creation and deletion of generic nodes, links and groups are supported. It is possible to define connections and groupings. The user of the system is able to move the graphical representation of objects on the screen at will.

The interface between the Modeler and the User Interface is developed using the Sun RPC/XDR interface protocol [6]. This interface allows the UI to invoke remote procedure calls to the Modeler to get services performed on its behalf. This interface was chosen for the prototype since it is easy to use, and appropriate at the present time. It is our intention to follow ISO ASN.1 standards for information exchange when appropriate tools to do so become more available. Until then, we will use two RPC interfaces for bi-directional communication between the Modeler and each of its clients.

The Modeler, implemented in VBase, currently has the generic Network Configuration. In this prototype version, it includes creation and deletion of nodes, links and groups, connections between nodes and links, and addition and deletion of simple and group objects to and from a given group. It also returns objects that a given node or link is connected to, children objects of a given group, parent objects of a given group or a simple object, and so on. As mentioned earlier, while VBase is not a distributed database, it has been fairly easy to exchange objects between the Modeler and the User Interface as 64-bit object ids defined in VBase. A preliminary design for the User Interface information to be stored in the Organization model of the Modeler is complete.

For the second version of NetMATE, we have decided upon the structure of the Network Definition partition for an X.25 network and the interface between the User Interface and the Modeler that allows the User Interface to inquire about the specific properties of nodes and links, and their values. Our next goal is to define the same for a TCP/IP local area network and attach the NetMATE system to a physical network.

7 Conclusion

In today's heterogeneous network environment, network management is a challenging and complicated task. Our preliminary design and implementation results for NetMATE have demonstrated the utility and flexibility of an object-oriented approach to model not only a network but also the network management and organization structures. The modular development of NetMATE will allow a distributed, efficient and portable solution to manage diverse networks, both real and simulated.

References

[1] Bacon D., Dupuy A., Schwartz J., Yemini Y., "NEST: A Network Simulation and Prototype Tool," Proceedings Usenix Conference Winter 87, Dallas, Texas.

[2] Dupuy A., Schwartz J.. "NeST System Overview," Technical Report CUCS-375-88, Department of Computer Science, Columbia University, New York, NY.

[3] Feridun M., Leib M., Nodine M., Ong J. "ANM: Automated Network Management System," IEEE Network, V2, N2, March 1988, pp 13-19.

[4] Klerer S.M. "The OSI Management Architecture: an Overview," IEEE Network, V2, N2, March 1988, pp 20-29.

[5] ISO TC97/SC21/WG4/N399 Management Framework Editing Meeting. "Position Paper Concerning OSI Management Domains," Tokyo, June 1987.

[6] Remote Procedure Call/External Data Representation, Network Programming, Sun Manual.

[7] Vbase Integrated Object System, Technical Notes, Ontologic Incorporated, Billerica, MA.

[8] Willett M., Martin R.D., "LAN Management in an IBM Framework," IEEE Network, V2, N2, March 1988, pp 6-12.

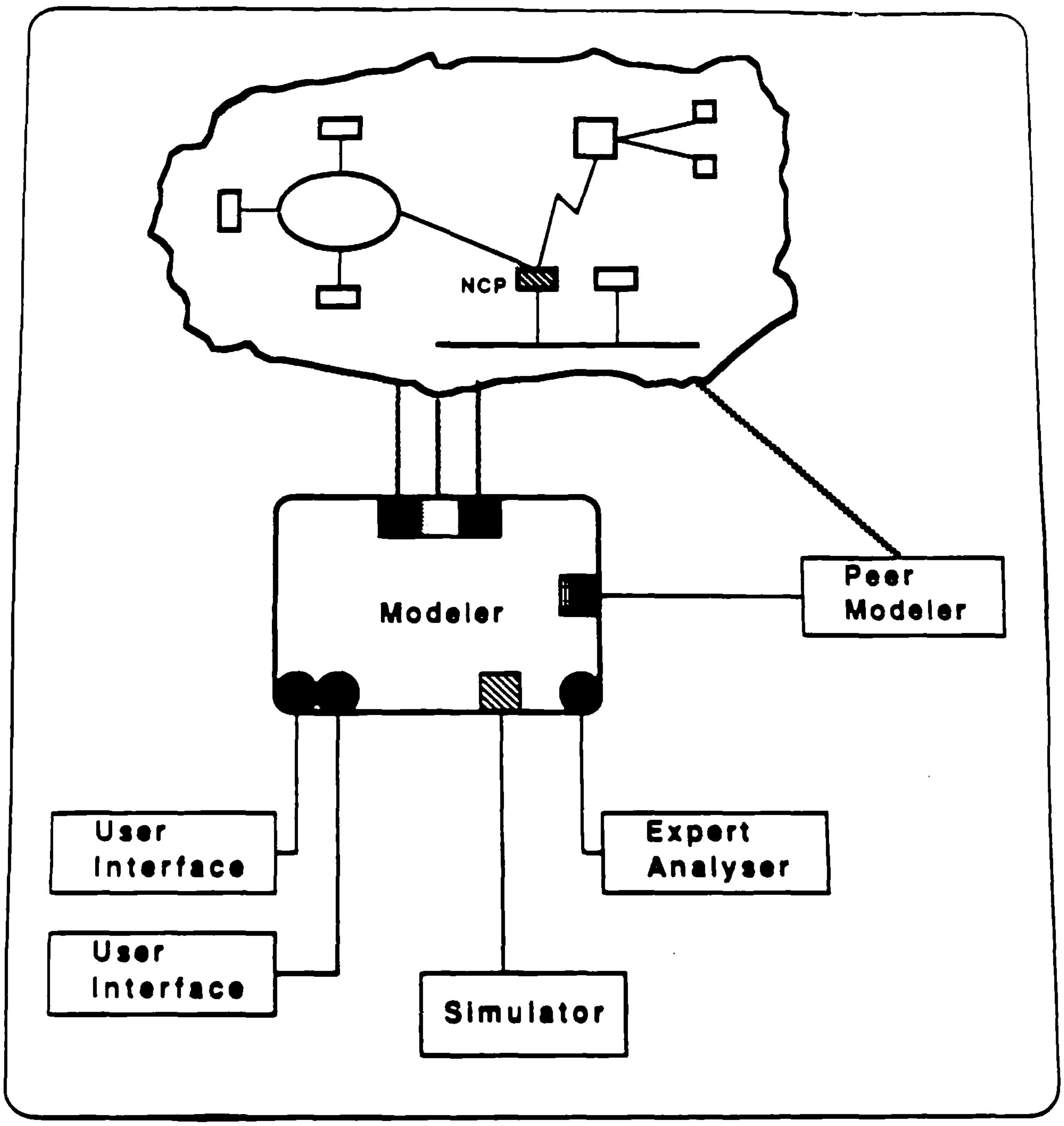

Figure 1. NetMATE Architecture

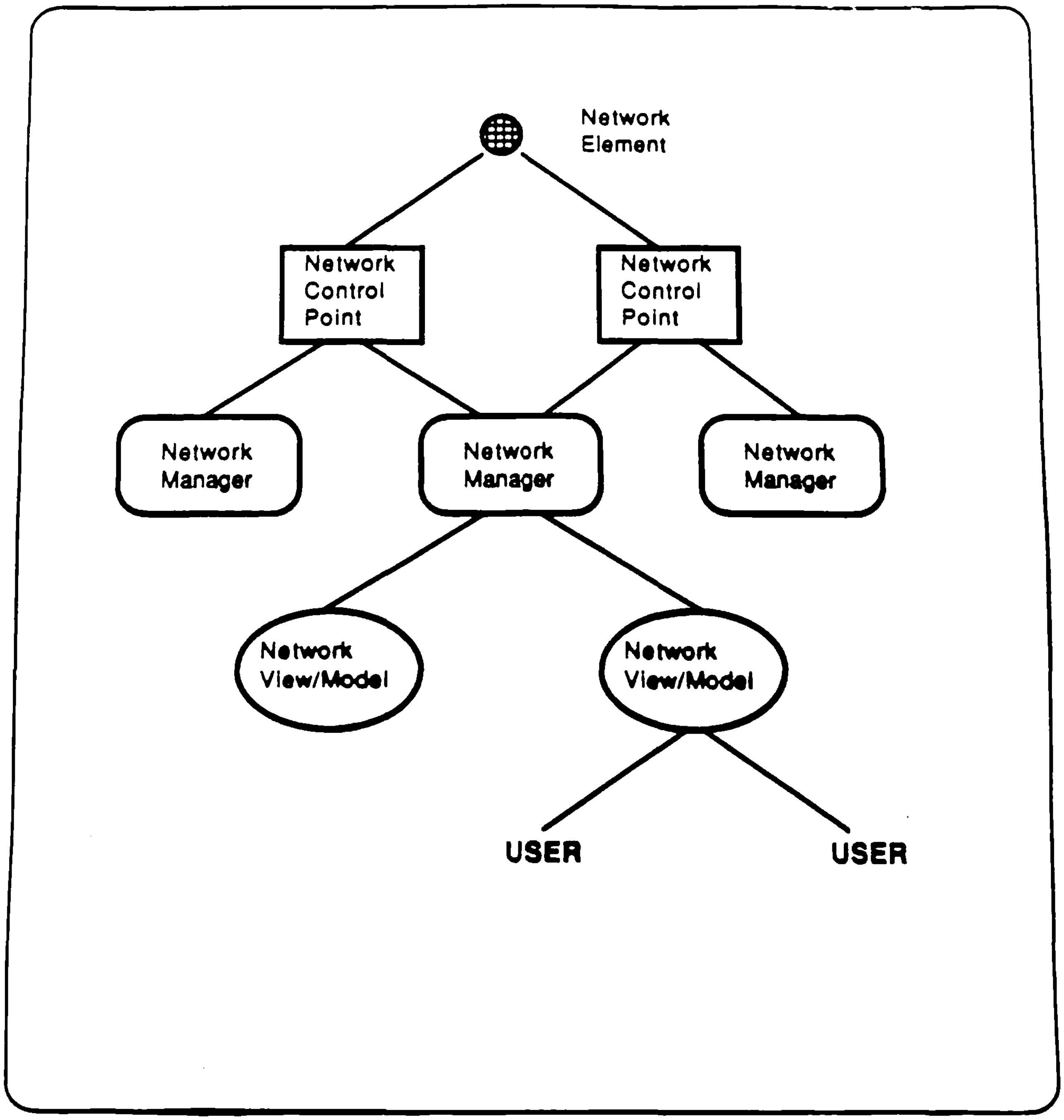

Figure 2. Network Management Control/Information Flow

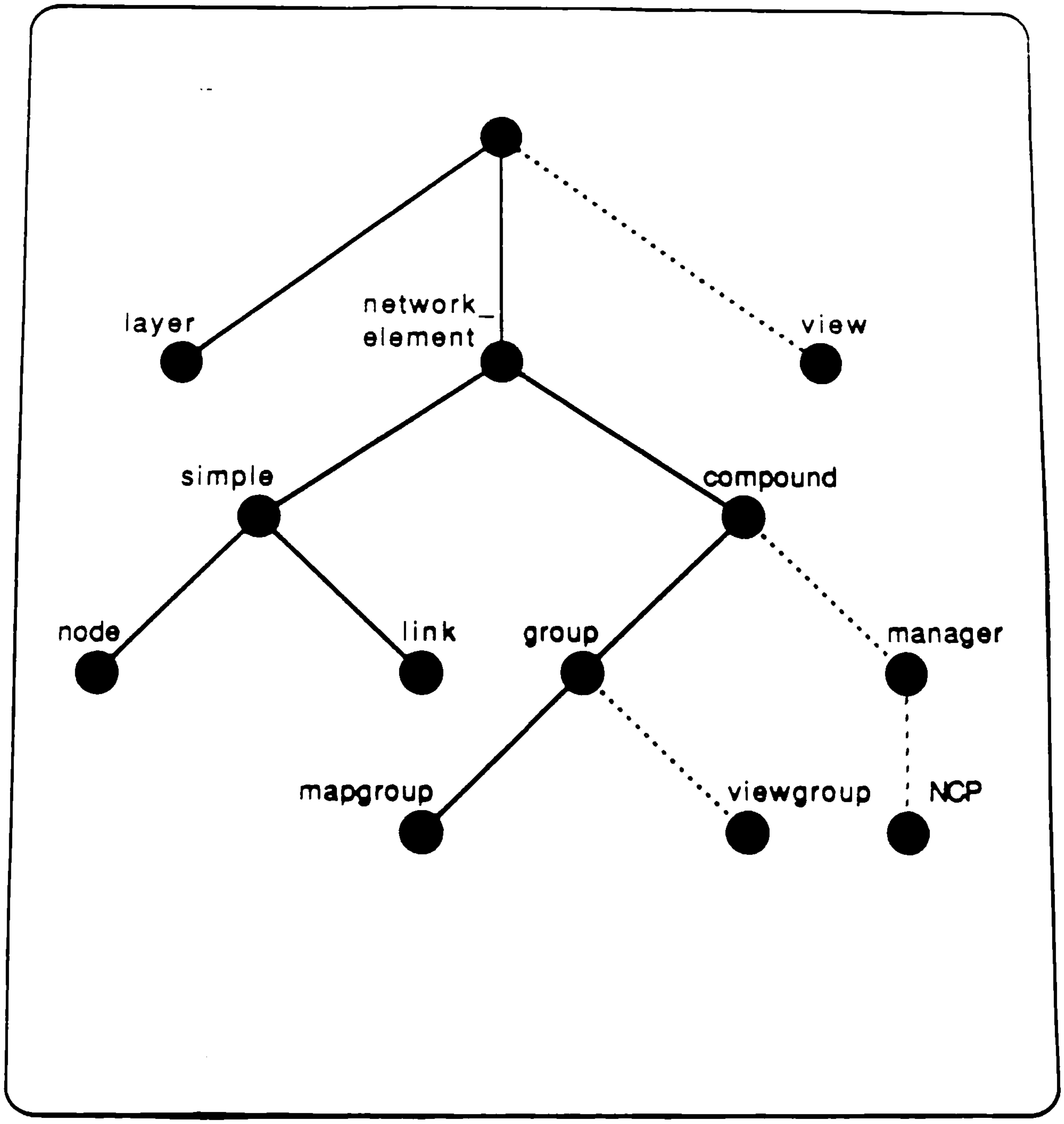

Figure 3. NetMATE Generic Type hierarchy

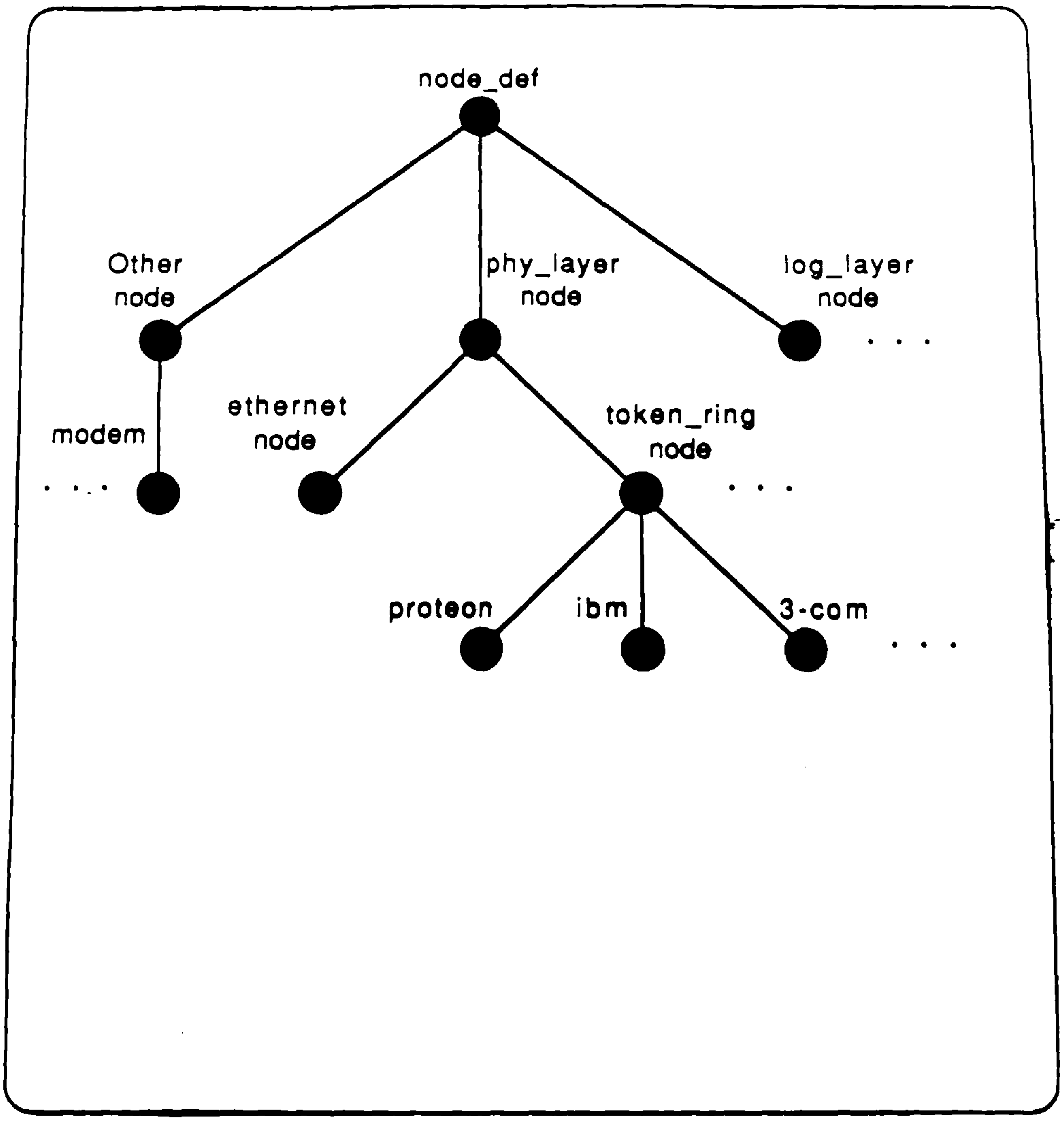

Figure 4. NetMATE Definition Type hierarchy (sample)

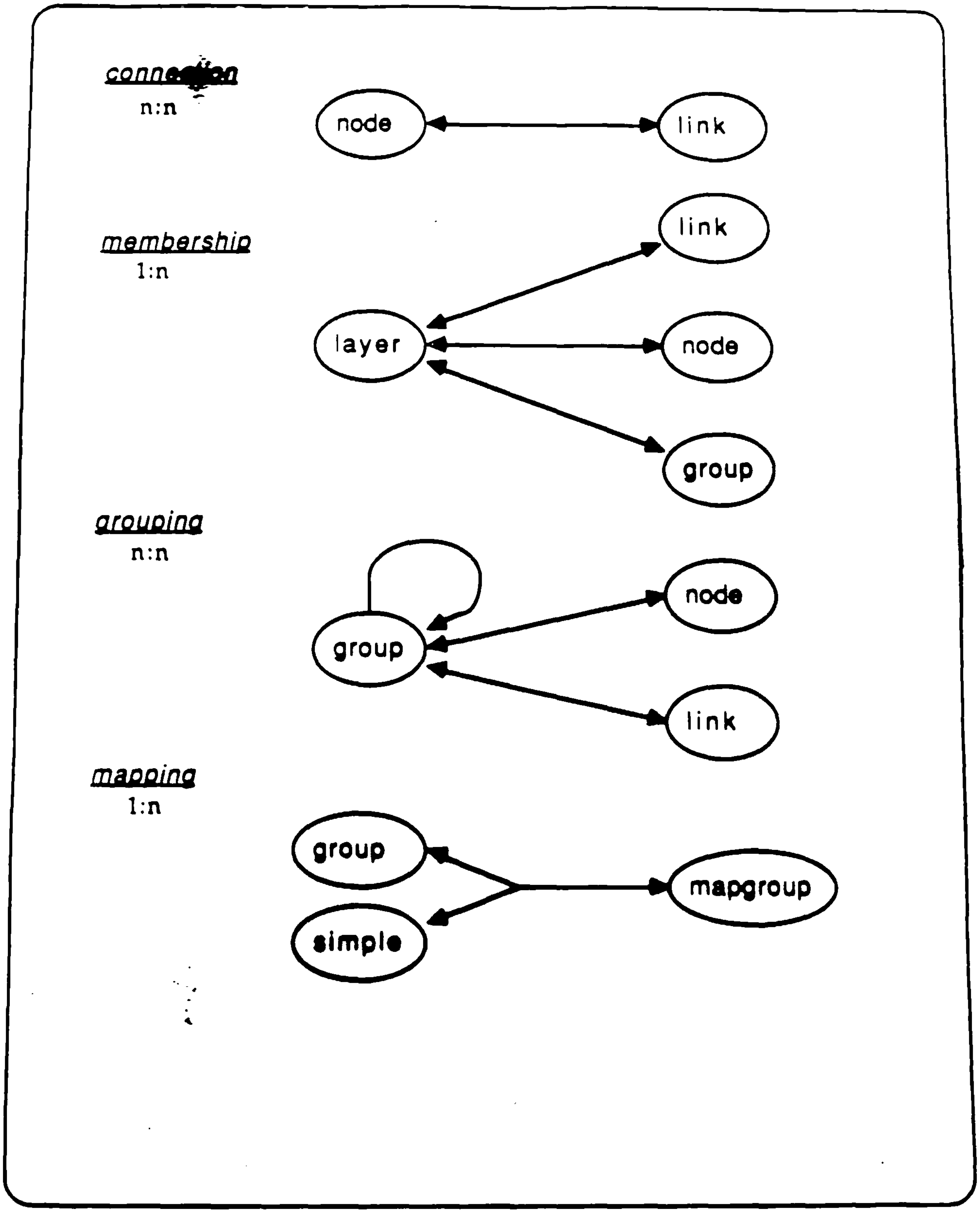

Figure 5. NetMATE Generic relations