The NetML Network Model

Phil Bohannon
Steven Fortune
Cliff Martin[1]

May 25, 2004

Abstract

NetML is a language for recording a snapshot of the physical and logical topology of a computer communications network. It is applicable across multiple network levels and to a wide variety of network technologies. Fundamental to NetML is a formal topology model that encompasses links, encapsulation, multiplexing, communication paths, and the binding of links to paths. NetML has an XML syntax and is described by an XML schema; this allows use of a wide variety of XML tools. This paper describes the NetML topology model, NetML syntax, and some preliminary experience with NetML and NetML-specific tools.

1 Introduction

Computer communication networks are extremely complex. One cause of complexity is the layered nature of networks, with a logical link at one technological level provided as a service by a different level. For example, a web browser process might establish a TCP connection to a server process; the connection appears to the two processes as a link. In fact, data sent across the connection traverses an underlying connectionless IP network, in which each link might be provided by a complex hierarchy of connection-oriented ATM, SONET, and DWDM networks. This complexity is useful—the internet exists in part because IP can be routed over almost anything—but the complexity challenges network analysis. For example, the question "why is a link slow?" may only have an answer that involves multiple network levels. The layering complexity of networks can only be expected to increase in the future, with the advent of wireless links, virtual private networks, overlaid protocols like SIP, etc.

Software tools play a critical role in designing, inventorying, analyzing, and optimizing networks. Fundamental to any such tool is a model of network topology; that is, the structure of links and connections in the network. Somewhat surprisingly, there appears to be no widely-used model that can uniformly describe networks across multiple levels and technologies. The seven-layer ISO model and variants [12] provide a useful conceptual understanding of networks, but they are too abstract to provide a concrete representation of a specific network. For purposes of visualization, a network is often modeled simply as a graph; however, a graph is inadequate since it does not describe how links are formed from other links.

Existing network tools seem to fall into two categories. Some tools are specific to a particular network technology. A network technology is usually defined by a standards document, which is typically very detailed and hard to abstract. In consequence the network model used by a technology-specific tool also can be very detailed and hard to abstract. Other tools operate at a higher level; for example a path-planning tool may route demands given by a demand matrix through a network given as a graph. However, the underlying model is often ad hoc, and the interpretation of, say, a 'node' in one tool may not match the the interpretation in another.

[1] Bell Laboratories, Lucent Technologies, 600 Mountain Ave, Murray Hill, NJ 07974. {bohannon|sjf|cliff}@lucent.com.

In a multitechnology world, it is imperative that network tools interoperate; in the absence of a common network model, interoperation is difficult. For example, data interchange between two tools requires a careful semantic analysis of the underlying models, followed by a conversion from one model to another. Similarly, it may not be clear whether an existing tool designed for one technology can be reused for another; indeed it is unclear even how to design a new tool to work across technologies.

This paper describes a multi-layer topology model for networks. The model is intended to describe a snapshot of an existing or potential network. The model encompasses links, adaptation, connections, communication paths, and binding. Links are given types, called layers, that characterize the format of data. Adaptation models both encapsulation, the changing of a data stream from one format to another, and multiplexing, the combination of several streams of data into one. Links can be connected together to form communication paths, possibly using adaptation. A link can be bound to a communication path, that is, defined by the path. This binding allows links to be viewed hierarchically, with one link defined in terms of tributaries of other links. The model is intended to be simple, applicable to both connection-oriented and connectionless networks, and independent of any particular technology. Specific technologies are described by choosing appropriate layers and adapters; section 3 below describes the layers and adapters needed to model ATM, SONET, DWDM, IP and MAC.

The topology model described in this paper is the foundation of NetML, a concrete XML dialect for data exchange among tools that manipulate network snapshots. This dialect, discussed in Section 4, provides naming, a containment hierarchy, and primitives representing link, network elements, ports, and so forth. Adaptation is expressed using port hierarchies. NetML is defined in part using XML schema [14, 7]; this provides a relatively simple way to enforce syntactic integrity constraints on a NetML instance. The schema is designed so that applications can augment NetML files with additional information, while still maintaining schema compliance. Section 5 discusses NetML-specific tools, including programming interfaces, visualization, and a special-purpose validator that checks semantic integrity constraints.

NetML is currently being used for two network analysis applications. NetML files are produced by a tool[5] that traverses an IP network, querying each host and router for its local state and its links to other network elements. The resulting topology can be visualized by a graph-drawing tool that reads NetML files. NetML has also been used to provide an intermediate network description for a process that optimizes SONET circuit placement. Section 6 briefly describes these early experiences with NetML.

There are a wide variety of potential applications for NetML as a concrete exchange format. For example, NetML files could be read and written by a management system that has global knowledge of a network, or by a design tool that designs a network. NetML files could be analyzed and visualized in numerous ways, e.g. to show link utilization, produce traffic summaries, reconcile different network data bases, plan circuit rearrangement, analyze alarms, show recent changes, show unexpected links, estimate reliability, etc. We believe that NetML is an outstanding foundation for the development of network tools of unprecedented generality and flexibility.

Contributions of the model. Various network topology models have appeared in industrial standards documents. For example, ITU G.805 [9, 15] considers connection-oriented network elements, such as SONET and SDH; the TMF 513 model [13] somewhat simplifies the G.805 model. ITU G.809 [10] considers connectionless networks.

The NetML topology model certainly borrows ideas from these prior models, for example, the terms “layer” and “adaptation.” The NetML model has several novel contributions. First, the NetML model has a precise and reasonably compact mathematical description, which does not depend upon any particular network technology. Second, the NetML model uniformly incorporates packet-switched networks like IP and ethernet, allowing descriptions of communication paths that traverse both packet- and circuit-switched networks. Third, NetML has a concrete XML representation, allowing interchange of network descriptions among computer tools. Finally, the NetML model is simpler and more abstract than the prior models. The TMF 513 model appears to be intimately tied to a particular hierarchical management structure of networks. For example, the terms “link connection”, “subnetwork connection”, “trail”, and “topological link” all appear to be links but at different points in the hierarchy. NetML has a uniform notion of link independent of network hierarchy.

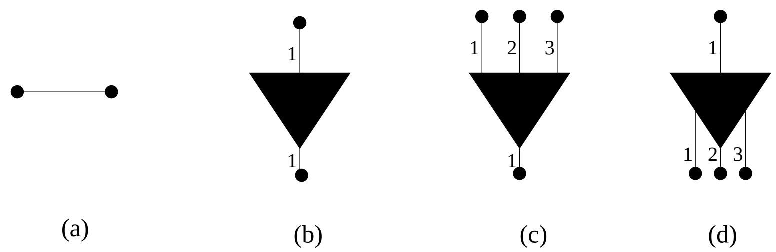

Figure 1: (a) a link and its two incident ports; (b) an adapter with a single guest port adapting to a single host port; (c) an adapter that multiplexes three guest ports to a single host port; (d) an adapter that reverse multiplexes one guest port to three host ports.

2 The formal topology model

This section describes the topology model abstractly, in a way largely independent of any particular network technology. Section 3 below shows how the model can be applied to several standard technologies.

Formally a *network* is a set of ports, a set of links, a set of adapters, and two binary relations on ports, one for connections and one for bindings. A link is a pair of ports; an adapter is a set of at least two ports. Each port appears in exactly one link or adapter. The two relations satisfy conditions as described below.

2.1 Ports, links, layers, and adaptation

A *link* is the ability to transmit data from one place to another. A link has exactly two ports, the *start* port and the *end* port. A link can be interpreted operationally: data injected into one port traverses the link and is then ejected at the other port. A link is bidirectional, so data injected at either port will be ejected at the other port. A link is drawn as a line segment, with ports as small dots; see figure 1(a).

Links and ports can be interpreted concretely, as physical objects. For example, a link could be a copper or fiber cable and a port could be the physical connector at the end of the cable. More generally, a link could be a wireless connection and the port might be some connector attached to antenna circuitry. Links and ports can also be interpreted more abstractly. For example, a port might correspond to a subset of the data stream present at a physical port that can be independently switched by a network element. Similarly, a link might correspond to several physical links connected in sequence.

A link or port is characterized by its layer. A *layer* is the relevant set of information required to describe the data flowing on a link. The layer could include bandwidth, bit encoding scheme, error correcting codes, signaling protocol, framing format, header information, etc.; a layer is always viewed as an aggregate and not decomposed. A link and its endpoint ports always have the same layer.

Adaptation converts between layers. Adaptation represents both encapsulation, the encoding of one data stream within another, for example by adding extra header information, and multiplexing, where several data streams are combined into a single stream.

An *adapter* performs adaptation. An adapter has an indexed set of *guest* ports and an indexed set of *host* ports. Consider an adapter with a single guest port and a single host port. Any data stream of the appropriate layer can be injected into the guest port, where it is adapted to the layer of the host port and then ejected. An adapter is bidirectional, so it can be used to convert back from the host layer to the guest layer. However, it is only meaningful to inject a previously adapted data stream into a host port. Adapters are given types, see below, to ensure consistent adaptation and unadaption. Adapters are drawn as triangles, usually pointing downwards, with guest ports on the top flat side and host ports on the bottom pointed side. The index labeling a guest or host port is drawn on the side of the line segment connecting the port to the adapter triangle; if there is a unique host or guest port, the label is sometimes omitted. See figure 1(b)-(d).

An adapter with multiple guest ports and a single host port models multiplexing: a number of data

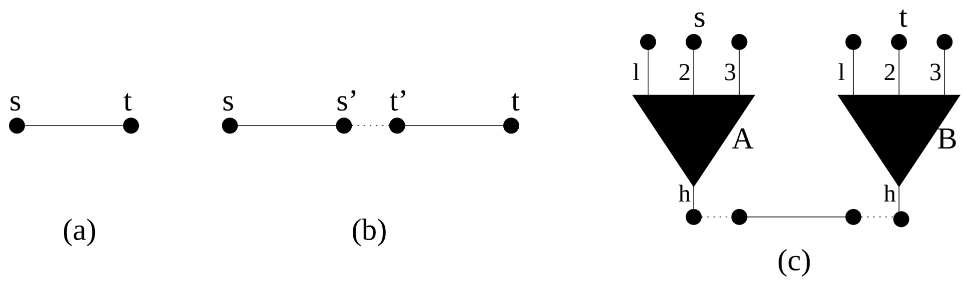

Figure 2: Three cases for a communication path from s to t: (a) a link; (b) s and s' communicate, t' and t communicate, and s' and t' are connected; (c) s and t are guest ports with the same label of two adapters A and B of the same type, and the host ports of A and B are connected to a pair of communicating ports.

streams can be combined together into a single layer. An adapter with a single host port and several guest ports models reverse multiplexing: an arbitrary data stream can be split into several data streams. Unadaptation of reverse multiplexing would recombine the data streams together. (Reverse multiplexing is relatively rare, but occasionally a single high-bandwidth data stream is split into several low-bandwidth streams for transport.)

To maintain consistency of adaptation and unadaptation, each adapter has a type defining the label and layer of ports associated with the adapter. In particular, an adapter type has a *host function* mapping *host labels* to layers, and a similar *guest function*. For each host label, the adapter must have a host port with that label and with layer determined by the host function. For each guest label, the adapter may optionally have a guest port with that label and with layer determined by the guest function. Guest and host labels must be disjoint.

2.2 Connections and paths

A port can be *connected* to another port at the same layer. Data ejected from one port is injected to the other. Connection is symmetric: if one port is connected to a second port, then the second is connected to the first. Connection is antireflexive: a port is never connected to itself. Ignoring bindings (section 2.3), connection is almost transitive: if one port is connected to a second, and the second to a third, and the first is distinct from the third, then the first and third are connected. If several ports are connected, then any data ejected from one port is injected into all the others. A port that is not connected is a *free* port. For the remainder of section 2, a connection is depicted by a horizontal dotted line (see figure 2(b)).

Connecting two ports can be interpreted concretely as plugging two physical connectors together. The interpretation can also be abstract; for example, a connection could be the result of a command to a network element to route a particular tributary of one physical port to a tributary of another physical port.

There is a *communication path* between two ports s and t, equivalently s and t *communicate*, if

1. s and t are the endpoint ports of a link, or
2. there are connected ports s' and t' with s and s' communicating and t' and t communicating, or
3. s and t are guest ports of two adapters A and B of the same type, s and t have matching labels, and for each pair of host ports a and b of A and B, respectively, with the same label, a is connected to a port a' that communicates with a port b' connected to b.

See figure 2. For the moment only identical labels match; a different notion of matching is used in section 2.4 below. Clearly, if there is a communication path from port s to port t, then any data injected into s will be emitted from t.

If a set of links is connected to form a tree, then every pair of leaf ports communicate by a unique path. A set of links could also be connected to form a graph with cycles, and there might be two ports connected by two different paths. Data injected into one of the ports will be ejected from the other; the operational interpretation is that the data will traverse one of the paths, chosen in an unspecified way (and not that the data will loop forever within a cycle).

Free ports have two interpretations. *Internal* free ports model connections that could be made but have not been made, e.g. a physical connector that has not been plugged into another connector, or a logical port corresponding to an unprovisioned time slot. *External* free ports indicate points at which a data stream that is external to the model could be injected or ejected. An example might be a port of a server network that in fact is connected to a client network, but the client network is not modeled.

2.3 Binding

Binding provides a way to define a link in terms of a communication path. Binding is a relation on pairs of ports; a pair (s,t) can appear in the relation if s and t have the same layer, and s is the end port of a link; s is the *bound* port with *binding* t. A link is *bound* if either of its endpoint ports are bound. Informally, a bound link is *valid* if both its endpoint ports are bound, and there is a communication path between the bindings of its endpoint ports.

A valid bound link is just a name for a communication path. A bound link in part models a "provisioned circuit" common in circuit-switched technologies like SONET. An endpoint port of a bound link can be connected to another link; however this connection must be consistent with the connection of the binding of the port, as described below.

Binding can be nested, allowing a hierarchical description of links (see figure 3(a)). In a complete description of a network, the bottommost links would be *physical* links, for example, corresponding to copper or optical cables. However, the description need not be complete. The bottommost links could be *abstract* links, that is, links whose definition (either as a physical link or in terms of other links) is not known.

A little care is needed to formally define binding and relate binding and connections. The binding relation must be a partial function, i.e. a port can have at most one binding (though two ports can have the same binding). Binding must be acyclic, i.e. for any port p, the sequence of ports obtained by taking successive bindings must eventually reach a port with no binding; this last port is the *root binding* of p. A link is *bound* if either of its endpoint ports are bound. A bound link is *valid* if both its endpoint ports are bound and there is a communication path between the root bindings of its endpoint ports, where the path must use no bound links. See figure 3(a).

Connection must distribute over binding: if port b is bound to port c, and a is connected to either b or c, then a is connected to both b and c. This distributivity property guarantees that the two superficially different situations in figure 3(b) and (c) are in fact the same. Figure 3(b) is drawn with link l connected to link m, which is bound to a communication path formed by links p and q. Figure 3(c) is drawn with link l connected to link p, which is connected to link q, and m is bound to the path formed by p and q. By distributivity, figures 3(b) and (c) are the same.

A *bound group* of ports with *root* p is the set of all ports with root binding p. Restated, the distributivity condition says that connection applies to bound groups of ports: if any port in one bound group is connected to any port in another, then every port in the first group is connected to every port in the second group.

Connection is transitive modulo bound groups: if a port a in one bound group is connected to a port b in a second bound group, and b is connected to a port c in a distinct third bound group, then a is connected to c (and every port in the first group is connected to every port in the third group).

2.4 Circuit and packet switching

The model as described so far does not directly imply the ability to determine the source of a data stream. For example, if a set of links is connected to form a tree, then data ejected at a leaf port could have been injected

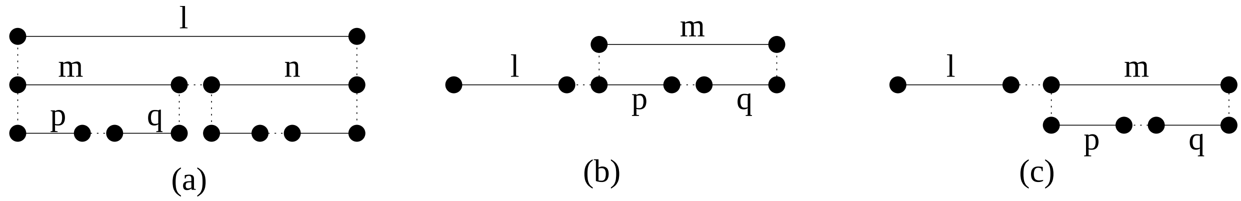

Figure 3: Vertical dotted lines indicate binding; horizontal dotted lines indicate connections. In (a) link l is bound to the path formed by links m and n; similarly m is bound to the path formed by links p and q. The distribution law for connectivity implies that that (b) and (c) are the same situation, i.e. in both cases the right ending port of l is connected to the left ending port of both m and p. Similarly, in (a), the right ending port of q is connected to the left ending port of n.

at any other leaf port. Two classes of layers, circuit-switching and packet-switching, model connection-oriented and connectionless networks, respectively. These two layer types impose additional restrictions on connections and communication paths; these restrictions guarantee that the source of a data stream can be determined. Both layer types can be intermixed in a single network description, in the sense that a circuit-switched communication path might be formed using adapted packet-switched links, and conversely.

Informally, a *circuit-switched* layer imposes the additional restriction that at most two ports of that layer can be connected together. Formally, the restriction is that at most two bound groups of ports can be connected together. Ignoring bindings, this restriction implies a set of links can only be connected to form either a path or a cycle. Adaptation to a circuit-switched layer is as described above. A communication path connecting two ports, if it exists, is unique as long as all adaptations are to circuit-switched layers.

A *packet-switched* layer has an associated universe of *addresses* and models routing with source and destination addresses chosen from the universe. A packet-switched layer allows arbitrarily many ports to be connected together. However, adaptation to a packet-switched layer is restricted: each adapter to a packet-switched layer must have only a single host port. Each adapter guest port is labeled with a pair (l, r) of addresses, where l is the *local* address and r the *remote* address. An adapter guest port label (l, r) *matches* a guest port label (r, l) at a different adapter. Restating the definition in section 2.2, there is a communication path from s to t if

3. s and t are guest ports of two adapters A_s and A_t of the same type, s has label (l, r), t has label (r, l), for some addresses r and l, and the host ports of A_s and A_t communicate.

Given s, the corresponding port t must be unique if it exists, that is, there should be at most one adapter A_t so that the host ports of A_s and A_t communicate and A_t has a guest port labeled (r, l).

The communication path from s to t can be interpreted operationally as follows. Adapters A_s and A_t have addresses l and r, respectively. Data injected into port s is split into packets by adapter A_s, with header information giving r as the destination address and l as the source address. The packetized data traverses the path from the host port of A_s to the host port of A_t. The adapter A_t notices that the destination address is r and the source address is l; it removes the header information and ejects the data from port t. The path from t to s is similar; the label (r, l) at t now specifies that r and l are the destination and source addresses, respectively, for the header.

3 Examples

This section illustrates how various standard network technologies can be modeled using the topology model of section 2. Modeling a particular technology requires a choice of layers and allowed adaptations between layers. Then a specific network, with network elements and links, can be described using these layers and

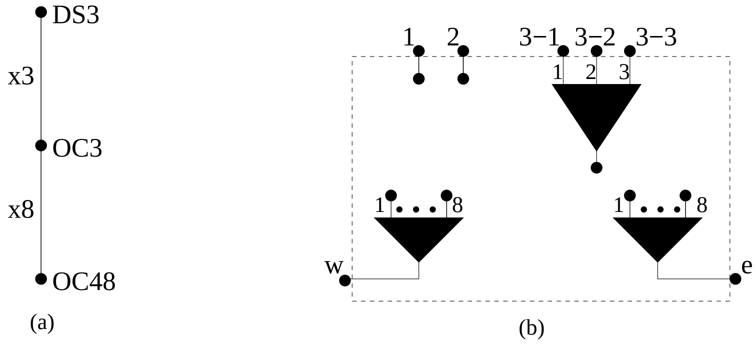

Figure 4: (a) Multiplexing hierarchy for the Sonet example (DS3 is 45 Mb/s, OC3 is 155 Mb/s, OC48 is 2.5 Gb/s); (b) The dotted rectangle delimits the add-drop terminal, with physical ports outside and logical ports inside. The physical ports include two OC48 ports (e and w), two OC3 ports (1 and 2), and a group4c of three physical DS1 ports (3-1 through 3-3) that are multiplexed to form an OC3 logical port. All logical ports are OC3.

adaptations. In general there are many possible ways to choose layers, representing a wide spectrum of possible levels of detail; in the examples below the number of layers is minimal for simplicity.

Examples are given by figures, e.g., figures 4 and 5. Henceforth connected ports are shown drawn one atop the other. Bindings between ports are shown with vertical dotted lines. Network elements are indicated with dashed lines; note that these are not part of the formal model of section 2, but are part of the concrete XML representation discussed in section 4. A physical port of a network element is just outside the outline; logical ports are inside.

3.1 SONET and SDH

Models of Sonet and SDH equipment have been extensively discussed in industrial standards documents [8, 13, 9, 15]. This section illustrates how the NetML topology model can capture the physical and logical connectivity of such equipment. A ring of add-drop terminals is used as an example; the terminal described here is loosely based on the Lucent FT2000.

An add-drop terminal has two physical OC48 ports, east and west, and several additional slots. Each slot can contain either a card with a single OC3 port or a card with three DS3 ports. Internally, the three DS3 ports are multiplexed together to form a single (logical) OC3 port; switching at the DS3 layer is not possible. The data at an OC48 port consists of 8 OC3 time slots (the other 8 OC3 time slots are used for protection, not modeled here). Each OC3 time slot can be either ignored, passed through to the same time slot on the other OC48 port, or routed to any logical OC3 port, corresponding to a physical OC3 port or three DS3 ports. (Traffic is bidirectional, so traffic from an OC48 port dropped to a low-speed port is added in the opposite direction from the low-speed port to the OC48 port.)

A simple NetML model has three circuit-switched layers DS3, OC3, and OC48, with adaptation multiplexing given in figure 4(a). A possible terminal with no internal cross-connects is given in figure 4(b). Physical ports e and w are at layerOC48; physical ports 1 and 2 are OC3; physical ports 3a through 4c are DS1. The model in figure 4(b) shows immediate demultiplexing of each OC48 port into logical OC3 ports, and similarly the multiplexing of three DS3 ports into a logical OC3 port.

Figure 5 shows several terminals connected together, with a link connecting the east OC48 port of one terminal to the west OC48 port of the next. (In a complete network the terminals would be arranged in a ring). Some cross-connects are shown; for example the multiplexed combination of ports 3-1 through 3-3 of terminal A are routed on time slot 8 of the OC48 link connecting terminals A and B. Link l is bound to the communication path connecting physical port 1 of terminal A to physical port 1 of terminal B; this link

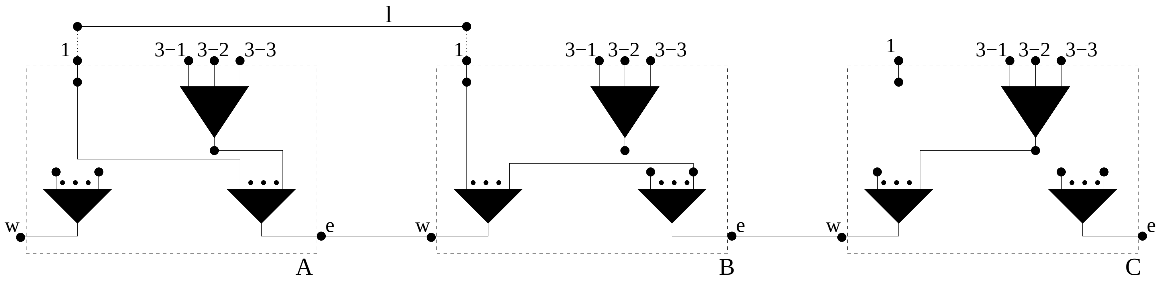

Figure 5: A network of add-drop terminals. Link l is bound to the communication path from port 1 of A to port 1 of B. There is a communication path from port 3-1 of A to port 3-1 of C, and similarly for ports 3-2 and 3-3.

could be an abstraction of the ring visible to a client of the ring.

A more realistic model of SONET/SDH equipment might have many more layers and adaptations. For example, TMF 513 [13] lists dozens of layers, and it is a simplification of ITU G805 [9]. Clearly the level of detail of modelling depends upon the needs of the application.

3.2 DWDM

Dense wave division multiplexing (DWDM) allows several different wavelengths of light, each independently modulated, to be combined on a single optical fiber. Modeling of DWDM network elements is similar to the modeling of SONET network elements, except that adaptation multiplexing now models multiplexing in wavelength rather than time.

A typical network element is an optical add-drop multiplexor (OADM). An OADM has two fibers incident; within the OADM the signal on the fiber is optically demultiplexed into individual wavelengths. Each wavelength can either be dropped, i.e. converted to an electrical signal and routed to a physical port, or passed through transparently to an optical multiplexor that combines all wavelengths together and sends the combination to the other fiber. As with the add-drop terminal, traffic is bidirectional.

Figure 4 requires only minor modification to represent an OADM. The layers would be OC48, OC192, and 'fiber', with four-one multiplexing from OC48 to OC192 and perhaps 128-one multiplexing from OC192 to fiber. The network element model in figure 4 would be essentially the same, up to multiplexing count.

3.3 ATM

The ATM protocol splits a data stream into cells, each cell consisting of data and header. The header includes a virtual channel identifier (VCI) and a virtual path identifier (VPI). An ATM switch routes an incoming ATM cell based on port identifier and the VCI and VPI of the cell. Each triple of incoming port, VCI, and VPI uniquely determines a triple consisting of outgoing port, new VCI, and new VPI; the ATM switch rewrites the cell using the new VCI and VPI and sends it out on the outgoing port. In order to save memory for lookup tables, an ATM switch can also route solely on the basis of the pair of incoming port and VPI. In this case the cell is rewritten with the new VPI but the VCI remains unchanged. ATM is connection-oriented (since outgoing port, VCI, and VPI are uniquely determined), hence can be modeled with circuit-switched layers in NetML.

Figure 6 depicts a possible adaptation hierarchy for ATM. Figure 6 models a detail of ATM: there are really two ATM cell formats, UNI (user-network interface) and NNI (network-network interface). The UNI format allocates 12 bits for the VPI, while NNI only allocates 8 bits; both formats allocate 16 bits for VCI. Adaptation to the virtual channel (VC) layer is done in two steps; this models the ability to route packets using just the VPI or both VPI and VCI.

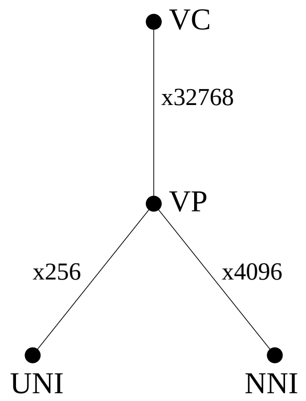

Figure 6: Possible ATM hierarchy; all layers are circuit-switched.

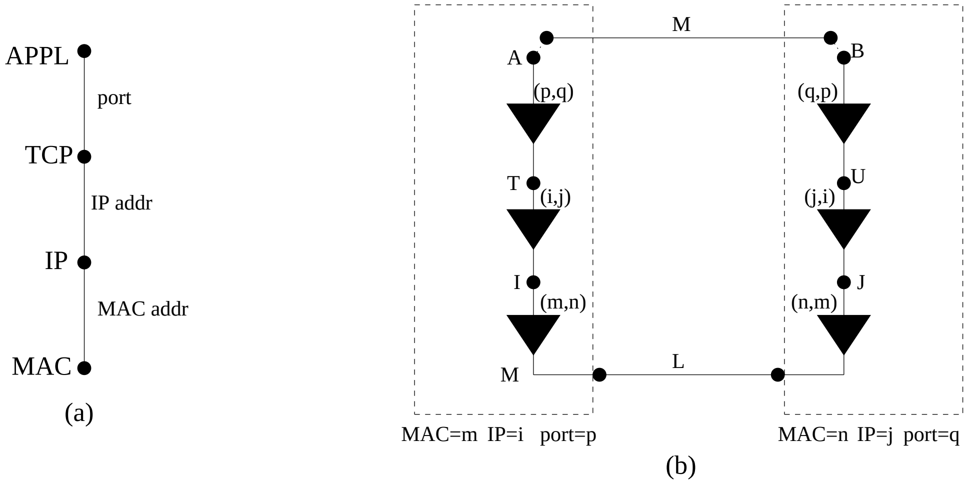

Figure 7: (a) MAC and IP adaptation hierarchy; (b) a pair of hosts.

Figure 4 with minor modifications applies to ATM. Each ATM port (either UNI or NNI) is the root of a two-level adaptation hierarchy. The top adaptation would be from VC to VP; guest ports would be labeled with VCIs. The bottom adaptation would be from VP to UNI or NNI; guest ports would be labeled with VPIs. Connections could be at the VC layer, corresponding to cell routing based on both VCI and VPI, or at the VP layer, corresponding to cell routing just on VPI.

3.4 MAC and IP

Modeling IP networks requires packet-switched layers. A simple model has three packet-switched layers, MAC, IP, and TCP, and a fourth circuit-switched layer APPL. The interpretation of MAC is a layer where the outermost packet headers are MAC headers, and similarly for IP and TCP. The APPL layer is a generic layer representing an interprocess data stream across a TCP connection.

The adaptation hierarchy is depicted in figure 7(a). Edge labels in the figure indicate the 'universe' of addresses; each adapter guest port is labeled with a local-remote pair of addresses. The number of potential guest port labels is huge. For example, an ethernet MAC address is six bytes, so the number of potential guest port labels on an IP-MAC adapter is 2^{96}.

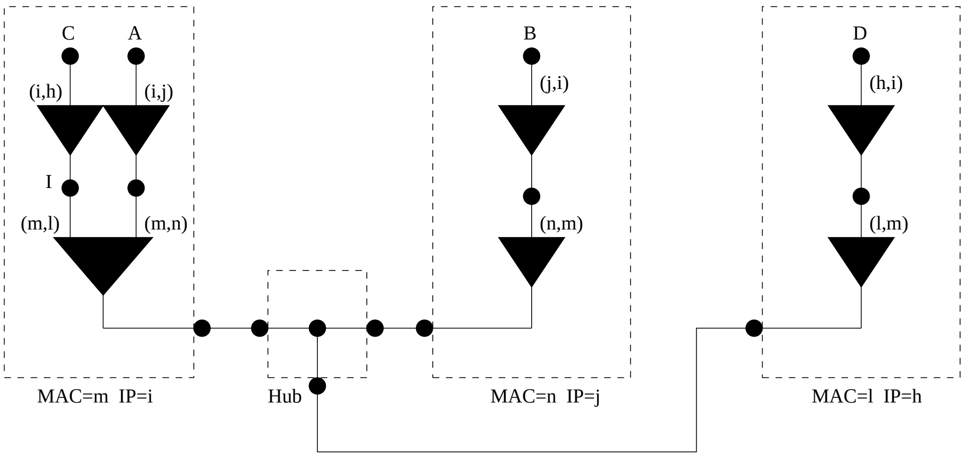

Figure 8: A network with three hosts and a hub. The layers from bottom to top are MAC, IP, and TCP. There is a communication path at the TCP layer from A to B, and also a path from C to D.

3.4.1 Hosts

Figure 7(b) depicts a simple IP network. There are two hosts connected by a MAC layer link L. The left host has MAC address m and IP address i; the right host has MAC address n and IP address j. Link L is an abstract link, in the sense of section 2.3, and could either be an ethernet cable or could be defined by some underlying transport network. The figure shows the adaptations required to obtain APPL layer ports A and B. According to the rules for communication paths in section 2.2, there is a communication path between A and B. Link M is bound to this path; link M models the link between a pair of application processes running on the two hosts. There is also a communication path between T and U, so a link could have been bound to the path; similarly for I and J.

The network model can be viewed operationally as follows. At port A, a process on the left host could inject application data. The first adapter adds TCP headers with source port p and destination port q. The next two adapters add IP and MAC headers with source-destination pairs (i, j) and (m, n), respectively. The data traverses link L, and goes through a sequence of adapters that remove the headers, and ends up at the port labeled B. A process on the right host could inject application data at port B, and the data would follow the reverse path to port A.

In general, there could be several communicating pairs of process on the two hosts, using different source-destination TCP port pairs. Each TCP port pair would label a different host port on the APPL-TCP adapter, and a distinct link at the APPL layer could be bound to the corresponding communication paths.

Figure 8 shows a more interesting situation with three hosts; the APPL layer is omitted for simplicity. The leftmost host communicates with two other hosts. Its IP-MAC adapter has two guest ports, labeled with local-remote MAC address pairs. The local address is the same for the two pairs; the remote addresses are the MAC addresses of the other hosts. Each guest port is further adapted to the TCP layer, using local-remote IP address pairs. There is a communication path at the TCP layer from A to B, as before, but now also a communication path from C to D.

3.4.2 Hubs, switches, and routers

It is possible to model other MAC and IP network elements. Figure 8 shows the model of a hub. A hub just broadcasts every MAC packet that arrives on some physical port to all other ports; it is modeled simply with a connection.

A MAC-layer switch also operates solely at the MAC layer, logically acting somewhat like a hub. A

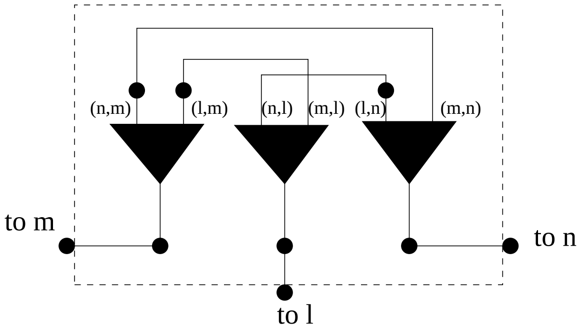

Figure 9: The internal model of a switch that might substitute for the hub in figure 8.

switch has several physical ports; it attempts to learn the MAC addresses of the network elements connected to each physical port. It then routes an incoming MAC packet to the correct outgoing physical port. A switch could be modeled like a hub; a more complete model is possible using adapters. Figure 9 gives the internal model of a switch that could substitute for the hub in figure 8. It has three physical ports, connected to the hosts with MAC addresses m, l, and n. It uses adapters to demultiplex based on MAC address pairs, and then uses connections to route based on the addresses. This switch model maintains communication at the IP layer; in other words, the communicating pairs of IP ports in figure 8 still communicate if figure 9 is substituted for the hub. The paths do change at the MAC layer, however.

Figure 9 is an example of a forwarding table, expressed using adapters. The total number of guest ports and connections is quadratic in the number of addresses (for each pair of addresses m and n, a guest port with label (m, n) is connected to a guest port with label (n, m).) Also, figure 9 is not really complete, since for example it does not describe what happens to a MAC packet with a new address. It would be reasonable to extend the network model with a primitive that expresses forwarding tables compactly; as figure 9 shows the meaning of the primitive can be defined in terms of adapters and connections, possibly many. However, this extension is not pursued here.

A router can be modeled much like a switch, except with routing based on IP addresses rather than MAC addresses; see figure 10(a). The physical ports are at the MAC level; each MAC level port is adapted up to the IP level. The dotted box within the router represents the forwarding table, which could be modeled simply as a connection, or with the more detailed model in figure 9.

3.4.3 Virtual private networks

The Bell Labs Viaduct box provides MAC-level virtual private networks. Viaduct boxes logically come in pairs. Each box has two MAC level ports, one private and one public. When the public ports are connected to a public IP network, the pair of Viaduct boxes automatically form a UDP connection. Any MAC level packet that is sent to a private port of one of the boxes is encrypted and packaged for transmission over the public IP network to the other box, where it is decrypted and sent out again as a MAC packet. To the private network, the pair of Viaduct boxes is invisible; the pair appears to be an ethernet link. To the public network, the pair of Viaduct boxes appears to be a pair of hosts.

The Viaduct boxes can be modeled as in figure 10(b); port numbers and IP and MAC addresses are omitted for simplicity. According to the rules for communication paths, the private MAC ports communicate. Thus a link can be bound to the path, as shown.

4 XML representation

NetML is expressed concretely in XML. Figure 11 shows the NetML representation of part of figure 5.

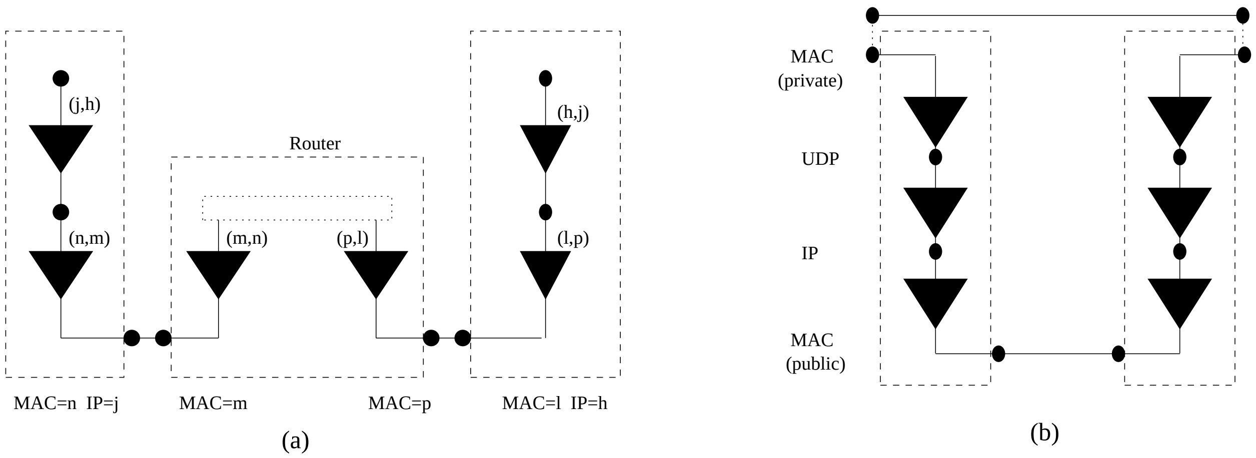

Figure 10: (a) A router between two hosts; (b) a virtual private network connection using Viaduct boxes.

A NetML file consists of a header and a body. The header contains a set of tables. The *layer table* enumerates the allowed layers, indicating for each layer whether it is circuit-switched or packet-switched. The *adapter class table* enumerates the allowed adapter types, determining for each type the count, labeling, and layer of guest and host ports for adapters of that type. Other optional tables include a location table, describing locations that can be used to tag various elements, and a source table that records the revision history of a NetML file.

The body of a NetML file consists of a set of networks. Networks are described using the XML elements depicted in figure 12. XML nesting indicates containment. Thus a `network` XML element is a container object that models a network; any XML elements that are its XML children are objects contained within the network.

XML nesting also determines naming. Every element in figure 12 (except `netML`) has an `id` attribute. The *local name* of an element is the XML element obtained from the element by deleting all its children and all its attributes except the `id` attribute. The *global name* of an element is the sequence of local names of the parent elements on the path from the root `netML` element to the element itself. For example,

```
<netML/><network id="MH"/><link id="L"/>
```

is the global name of the link in figure 11. This global name can be used as a reference to the element within the NetML file. A *relative name* is a sequence of local names, not starting with `<netML/>`. It specifies the element obtained by following the path from the `network` or `networkElement` that contains the occurrence of the relative name.

4.1 Networks and links

A `network` can contain `link`, `networkElement`, `connect`, and `bind` elements. A `link` element models a link as described in section 2. It has two endpoint ports, with `id` attribute values `start` and `end`, respectively. Similarly, the `connect` and `bind` elements correspond to the connect and bind operations in section 2. For example, within the network "SONET", the following

```
<connect>
   <ref><link id="l"/><port id="start"/></ref>
   <ref><networkElement id="2c501"/><port id="1"/></ref>
</connect>
```

```
<>
  <layerTable>
     <layer name="OC48" switching="circuit"/>
     <layer name="OC3" switching="circuit"/>
     <layer name="DS3" switching="circuit"/>
  </layerTable>
  <adapterClassTable>
     <adapterClass name="OC3-OC48"/>
         <host count="1" layer="OC48"/><guest count="8" layer="OC3"/>
     </adapterClass>
     <adapterClass name="DS3-OC3"/>
         <host count="1" layer=OC3"/><guest count="3" layer="DS3"/>
     </adapterClass>
  </adapterClassTable>
  <network id="SONET"/>
    <networkElement id="A">
      <port id="w" layer="OC48" adapterClass="OC3-OC48"/>
      <port id="e" layer="OC48" adapterClass="OC3-OC48">
          <port id="1" layer="OC3"/>
          <port id="8" layer="OC3"/>
      </port>
      <port id="3" layer="DS3-OC3">
          <port id="1" layer="DS3"/>
          <port id="2" layer="DS3"/>
          <port id="3" layer="DS3"/>
      </port>
      <connect> <ref><port id="1"/></ref> <ref><port id="e"/><port id="1"/></ref> </connect>
      <connect> <ref><port id="3"/></ref> <ref><port id="e"/><port id="8"/></ref> </connect>
    </networkElement>
    <networkElement id="B">
      <port id="w" layer="OC48" adapterClass="OC3-OC48"/>
          <port id="1" layer="OC3"/>
          <port id="8" layer="OC3"/>
      <port id="e" layer="OC48" adapterClass="OC3-OC48">
          <port id="8" layer="OC3"/>
      </port>
      <port id="3" layer="OC3"  adapterClass="DS3-OC3">
          <port id="1" layer="DS3"/>
          <port id="2" layer="DS3"/>
          <port id="3" layer="DS3"/>
      </port>
      <connect> <ref><port id="1"/></ref> <ref><port id="w"/><port id="1"/></ref> </connect>
      <connect> <ref><port id="w"/><port id="1"/></ref>  <ref><port id="e"/><port id="1"/></ref> </connect>
    </networkElement>
    <link id="ring1" layer="OC48">
       <start> <networkElement id="A"><port id="e"/></start>
       <end> <networkElement id="B"/><port id="w"/></end>
    </link>
    <link id="L" layer="OC3">
       <startbind> <networkElement id="A"><port id="1"/></start>
       <endbind> <networkElement id="B"/><port id="1"/></end>
    </link>
  </network> </netML>
```

Figure 11: A NetML file describing network elements A and B in figure 5.

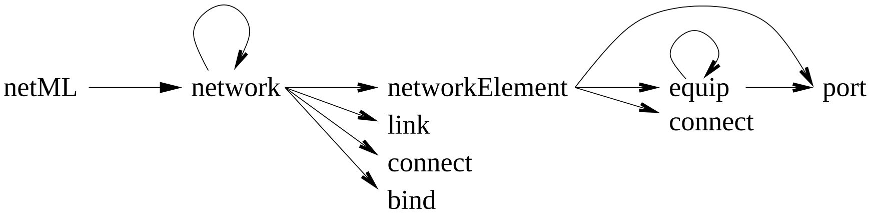

Figure 12: NetML element hierarchy. An arrow from one element to a second indicates that the first can contain zero or more instances of the second.

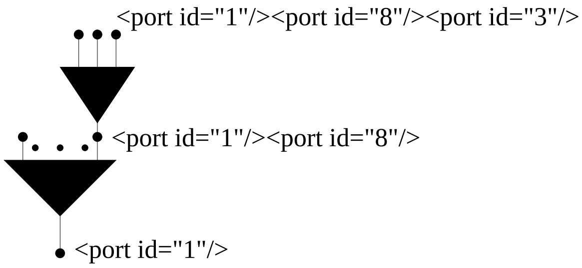

Figure 13: Hierarchical port naming; the bottom and top adapters are OC3-OC48 and DS3-OC3, respectively.

connects the start endpoint of link l to port 1 of network element 2c501. The connection and binding of link endpoint ports can be included in the definition of the link; thus

```
<link id="L" layer="OC48">
    <start><networkElement id="2c501"/><port id="1"/></start>
</link>
```

is equivalent to the link definition in figure 11 plus the `connect` just given.

4.2 Network elements

A `networkElement` is a container for `port`, `connect`, and `equip` elements. The `equip` element allows modeling of an equipment hierarchy (e.g. bays, shelves, cards). There seems to be no standard names for the levels of the hierarchy, so `equip` is used at every level. Equipment properties (see below) can be used to describe each level of the hierarchy.

While the `link` element defines a link, there is no corresponding element that directly defines an adapter. Instead, the `port` element defines a hierarchy of ports related by adaptation. Given the definitions of adapters and circuit-switched layers as in figure 11,

```
<port id="e" layer="OC48" adapterClass="OC3-OC48">
   <port id="1" layer="OC3"/>
   <port id="2" layer="OC3"/>
      ...
   <port id="8" layer="OC3"/>
</port>
```

defines a port with relative name `<port id="e"/>` and eight ports `<port id="e"/><port id="1"/>`, etc. with the former the host port of an adapter of type OC3-OC48 and the latter the guest ports. The guest port ids are the guest indices of the adapter of type OC3-OC48, in this case with numbers the default index set. Ports can be nested arbitrarily. For example, `<port id="8"/>` could be replaced with

```
<port id="8" layer="OC3" adapterClass="DS3-OC3"/>
   <port id="1" layer="DS3"/>
   <port id="2" layer="DS3"/>
   <port id="3" layer="DS3"/>
</port>
```

This would create three more ports `<port id="e"/><port id="8"/><port id="1"/>` etc., adapted from `<port id="e"/><port id="8"/>`. The resulting adapter hierarchy is shown in figure 13.

Packet-switched ports have extra attributes. Given packet-switched layers IP and MAC, the following

```
<port id="1" layer="MAC" adapterClass="IP-MAC">
    <port id="3" local="00:D0:59:BF:E4:26" remote="00:A0:C9:0A:C8:53" layer="IP"/>
    <address layer="MAC" local="00:D0:59:BF:E4:26"/>
    <address layer="IP" local="135.104.53.179"/>
</port>
```

defines a MAC port `<port id="1"/>` and an IP Port `<port id="1"/><port id="3"/>`. The latter can be interpreted as representing the data stream of IP packets sent to or from the MAC address 00:A0:C9:0A:C8:53. The optional `address` element serves two purposes. First, it labels a port with one or more addresses. Second, it constrains the `local` attribute of child ports. In this case, any child IP port of `<port id="1"/>` must have a local attribute matching the MAC address 00:D0:59:BF:E4:26; if `<port id="3"/>` had a further TCP child, its local attribute must match the IP address 135.104.53.179.

A `connect` element within a network element can be used to specify the connectivity of ports defined within the network element.

4.3 Equipment attributes

There are many attributes that can be used to describe network equipment. Typically these attributes are name-value pairs (e.g. the serial number is A04044234) and one might expect them to be expressed using the XML attribute syntax. However, the number of possible attributes is very large, though many are conceptually similar, and sometimes a given attributes can have several values.

NetML instead uses XML elements to describe attributes of network equipment. The XML elements express ‘structured name-value triples’, that is, triples consisting of a general property, a qualifier to the property, and the value. Each element has the general form

```
<property qualifier="qualifiervalue" value="val"/>.
```

Here `property` is an XML element name, and `qualifiervalue` is a string. A `networkElement` (or `equip` or `port`) can have an arbitrary number of these elements as children.

The current list of properties and qualifier values appears in figure 14, though this list can be expected to grow. For example, `code` is an alphanumeric string that completely characterizes the functionality of a card (network element, etc.); two cards with the same code are interchangeable. Unfortunately, there are several universes of codes, for historical or organizational reasons, and a card in general will have a characteristic code in each universe. Similarly `status` indicates whether or not a network element (card, port, etc.) is working correctly: the `operational` qualifier gives the overall status; the `administrative` qualifier gives commanded status by some operator or management system; other qualifiers break down the status in other ways. The `type` element gives the general functionality of a network element, in much coarser terms than `code`. The `physical` element with qualifier value `form` gives the form-factor of an object, e.g. whether the object is a card, shelf, bay, etc.

4.4 Extensibility

It is reasonable to expect that some applications will require attributes and elements not present in NetML. The NetML schema is designed to be extended in two ways.

property	qualifiervalue
code	CLEI ECI comcode apparatus
version	sw hw fw
physical	form
status	operational administrative major minor hw link secondary
type	function configuredFunction application

Figure 14: Equipment properties. The first column gives an XML element name; the second gives possible values of the `qualifier` attribute.

First, essentially every element is allowed to have a child `app` (application) element, with arbitrary XML content allowed underneath the `app` element. This mechanism is very simple, general, and allows easy addition of new content without first developing a new schema. However, it does not directly allow schema validation of the extended content. Second, essentially every element is allowed to have extra attributes and children, provided they come from a different namespace than the NetML namespace. (This is accomplished with the `any` and `anyAttribute` XML schema attributes, with `namespace` having the value `##other`.) The extra elements and attributes can be defined in a second schema; the reference to the schema can be placed at the beginning of the NetML file. Standard XML schema checking can validate the NetML content using the NetML schema, and the additional content using the second schema.

5 Tools

We have developed a few tools that are useful with NetML files. One tool displays a NetML file as a graph, drawing network elements as nodes and displaying links selected on the basis of layer. The tool uses a physics-based layout algorithm to choose node location, as necessary. A second tool is a network 'diff', reporting the differences between two NetML files.

We have experimented with various programming interfaces to NetML files. One is JAXB [11]. JAXB uses an XML schema to generate a set of JAVA classes that map very closely to the elements and attributes defined by the schema. An automatically-generated method reads an XML file and produces class instances corresponding to the file; similarly a JAXB method writes a file from a set of class instances. JAXB classes and methods are reasonably efficient, convenient to use, and guarantee schema-compliance. Unfortunately, the automatically-generated methods reflect only the syntax, not the semantics, of NetML.

A second programming interface, currently written in Python, embodies the semantic model of NetML. This interface simultaneously provides two views on the file. One view is based on the naming hierarchy; for example it is possible to traverse the naming tree from an object to its parent or to its children. The second view is the topology view, represented uniformly with ports, links, adapters, connections, and bindings. Every link appears in this view; every instance of a port hierarchy, as discussed in section 4.2, is replaced with a set of adapters, with appropriate connections between guest and host ports. Networks, network elements, equipment, etc. do not appear in the view, except implicitly to name ports and links. The basic primitive is to trace a communication path from a starting port; the primitive handles all details of adaptation and connections. This second interface is quite useful and powerful, though rather less efficient than the Java interface, since Python is an interpreted language.

A final tool performs semantic validation on NetML Files. There are many semantic properties of a NetML file that cannot be easily checked by XML schema validation. These includes ensuring that distinct objects have distinct names, that references are valid, that at most two circuit-switched ports are connected, and that layer information is consistent, e.g. any two connected ports have the same layer. More fundamental and complex to check is the validity of a binding; this requires tracing communication paths through the network model. A semantic validator, built on the Python interface, checks all these properties.

6 Applications

We have used NetML for two nontrivial applications, which we briefly describe.

6.1 Sonet ring optimization

Circuits on a Sonet ring are set up and torn down as traffic demands change. Over time, timeslot usage can become fragmented, causing inefficient use of capacity and perhaps preventing circuit setup even if adequate capacity is available. An important optimization reallocates circuits within and across rings to reduce fragmentation and even to free up entire rings. This optimization requires accurate knowledge of the circuits actually provisioned on each ring. The optimization algorithms are not trivial [1, 2] and are not discussed here.

NetML has been very useful as part of the data acquisition process that determines circuits. This process has three steps. First, the set of cross-connects in each terminal is obtained by a direct TL1 command to the terminal. This low-level access is more general and more likely to be accurate than querying a management system. The cross-connect report is given in a format designed primarily for human readability. The second step is to parse the reports and translate the cross-connect data into NetML. Finally, the NetML file is read and circuits extracted with a program that uses the NetML programming interface described above.

The data acquisition process could have been implemented without NetML. However, the process naturally splits into the two steps of parsing and circuit extraction; the interface between the two steps would require a data structure with the richness of NetML. Actually instantiating the cross-connect data as NetML allows uses of the NetML visualization tools. The NetML representation also forces modularity. For example, the circuit-extraction component is a bit complicated, handling partially-provisioned circuits, alarms, etc. With a NetML interface, it can be used with data cross-connect data obtained from other sources, as long as the data is converted to NetML.

6.2 Network inventory

Netinventory[5] is a tool that discovers the topology of an IP network. It traverses a proscribed subset of a network, querying routers and hosts to discover IP and MAC addresses as well as connections to other hosts and routers. It incorporates algorithms [3, 4] that infer ethernet (level 2) topology and express IP links (level 3) in terms of the ethernet topology.

Netinventory has recently been rewritten to use NetML as a common representation language. Hosts, routers, switches and hubs are represented using the model as described in section 3.4. Links are at two layers, Ethernet and IP. An ethernet link is abstract in the sense defined in section 2.3, that is, it may either be a physical ethernet cable or a logical link provided by a lower-level transport network. IP links are bound to communication paths constructed using adaptation over the Ethernet layer. In principle, of course, there could be an IP link between every pair of hosts. Instead, the convention is to report an IP link between a host and a router, but not between pairs of hosts.

NetML has provided a clean language for recording the two levels of topology and the relationship between them. The language uniformly extends to capturing ATM and frame relay networks, part of a planned extension to Netinventory. The validator has been helpful for debugging Netinventory algorithms. A file that fails to validate implies a bug somewhere—sometimes the bug is trivial, but in other cases the bug has been sufficiently subtle that it would have been missed without the validator.

7 Future directions

We believe that NetML is a clean, rich language expressive enough to capture the physical and logical topology of many kinds of networks. It should provide an excellent foundation for a suite of discovery, visualization, analysis, and optimization tools. There are a number of directions in which we expect NetML to continue to evolve.

Link hierarchies. NetML in some sense is quite low-level, describing ports, adaptations, connections, links, and network elements. A more abstract description that is sometimes appropriate is a link hierarchy. A link hierarchy explicitly defines each link as either abstract or as a connected sequence of tributaries of other links. A link hierarchy is implicit in a NetML representation; however, an explicit representation would be more convenient for some applications. The validator or another tool could automatically extract the hierarchy and augment the NetML file with the explicit representation.

Network design tools. A network design tool can be viewed abstractly as taking a partially specified network and specializing it, that is, adding information to make the network more completely specified. For example, routing a demand in a network can be viewed formally as providing a binding for a link (the demand); the binding is the routing of the demand, i.e. the path through the underlying network. Some suggestions for constructs that model partially specified networks are reported elsewhere [6].

Switching capabilities. Many SONET and SDH network elements have complicated constraints on their internal state, that is, what cross-connects are possible. An interesting and challenging extension to NetML would allow description of network element switching capability. Potentially, routing tools could take advantage of such a description to better provision circuits in SONET/SDH networks; similarly optimization tools such as described in section 6.1 could more accurately model switching capabilities.

References

[1] S. Acharya, B. Gupta, P. Risbood, A. Srivastava. Hitless engineering of SONET rings. To appear, *Globecom 2003*, San Francisco, Dec. 2003.

[2] S. Acharya, B. Gupta, P. Risbood, A. Srivastava. MöbiPack: Optimal hitless SONET defragmentation in near-optimal cost. To appear, *IEEE Infocom 2004*, Hong Kong, March 2004.

[3] Y. Bejerano, Y. Breitbart, M. Garofalakis, R. Rastogi. Physical topology discovery for large multi-subnet networks. *IEEE Infocom 2003.*

[4] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, A. Silberschatz. Topology discovery in heterogeneous IP networks. *IEEE Infocom 2000.*

[5] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, A. Silberschatz. Topology discovery in heterogeneous IP networks: the Netinventory system.

[6] S. Fortune. A simple formal model for network description. Manuscript, 2002. Available at `http://cm.bell-labs.com/who/sjf/pubs.html`.

[7] J. Ducket, O. Griffin, S. Mohr, F. Norton, I. Stokes-Reed, K. Williams, K. Gagle, N. Ozu, J. Tennison, *Professional XML Schemas*, Wrox press, Birmingham, 2001.

[8] *I-ETS 300 653: Telecommunications management network: Generic managed object class library for the network level view.* European Telecommunication Standards Institute. Available from `www.etsi.org`

[9] *ITU-T G.805: Generic functional architecture of transport networks*, International Telecommunication Union, Series G. Available from `www.itu.int`.

[10] *ITU-T G.809: Functional architecture of connectionless layer networks.*, International Telecommunication Union, Series G. Available from `www.itu.int`.

[11] Java architecture for XML binding. `http://java.sun.com/xml/jaxb/`.

[12] J. Kurose, K. Ross. *Computer Networking: a top-down approach featuring the Internet.* Addison-Wesley, New York, 2001.

[13] *TMF 513: Multitechnology network management business agreement, NML-EML interface*. TeleManagement Forum, May 2001. Available from `www.tmforum.org`.

[14] XML schema. Available at `http://www.w3.org/XML/Schema`.

[15] E. Varma *et al*, *Achieving Global Information Networking*, Artech House, Boston, 1999.