In *Proc. of the Network Operations and Management Symposium (NOMS 2002)*, Florence, Italy, April 2002.

Common Information vs. Information Overload

J. Schott and A.Westerinen
Cisco Systems
170 West Tasman Dr.
San Jose, CA 95134, USA
{jschott, andreaw}@cisco.com

J. P. Martin-Flatin
AT&T Labs Research
180 Park Avenue, Bldg. 103, Rm. A001
Florham Park, NJ 07932, USA
jp.martin-flatin@ieee.org

P. Rivera
LSI Logic
4420 ArrowsWest Dr
Colorado Springs, CO 80907
peter.rivera@lsil.com

Abstract

Coordinating management data across vendors and products is quickly becoming an overwhelming task. Sophisticated tools are needed to mine information from varying locations in MIBs, PIBs, CLI constructs, directory schemas, and other repositories and management standards. As data silos become larger and larger, and as individuals are expected to support more and more systems, the need for abstracted data that is consistent across vendors' platforms has become a requirement. The detailed information required for management and configuration must be abstracted into well-defined and meaningful concepts and terms. These concepts should cross vendor and product boundaries. This paper first describes the contributing factors to today's ineffective modeling efforts. It then proposes a new philosophy and approach for modeling management information, using relevant abstractions from the DMTF CIM Schema. Included in this work are recommendations to accomplish successful modeling of management data.

Keywords

Management, Information Modeling, CIM.

1. Introduction

The sheer quantity of management data that is available in the Internet world is staggering. As data silos become larger and larger, and as individuals are expected to support more and more systems, the need for abstracted data that uses consistent semantics across vendors' platforms has changed to a requirement. Many companies spend millions of dollars on management tools whose purpose is to mediate and normalize data between vendors and products. The best solution is for the products themselves to deliver information that is "mediated and normalized".

A second concern is the language of the management information [1]. It seems that each problem domain chooses its own terms for the same basic concepts. A case in point is the terminology for network interfaces. Various companies simply refer to *interfaces*. But the Distributed Management Task Force (DMTF) chooses the term *protocol endpoint* [2] while the International Telecommunication Union (ITU) uses the term *termination point* [3]. None of these terms are incorrect, but it is critical to standardize on one set and build upon it. The set of terms must be semantically explicit, but sufficiently abstracted to be usable and reusable.

Abstracted and reusable information facilitates improved management. Detailed information for configuration can be abstracted into the concepts useful to a network administrator (users, systems, ports, access points, etc.). One should not have to understand every minute detail, of each bit, in every layer of the protocol stack, to do network management. Standards bodies have worked hard to standardize on every bit in order for vendors' equipment to interoperate on the wire. This level of information is not manageable, but simply exists. High-level concepts like customers and ports must be managed. The reuse of these concepts is powerful. It enables implementation code and knowledge to be reused, which reduces design and development time.

This paper addresses the issues outlined above by first describing the contributing factors to today's ineffective modeling efforts. It then proposes a new philosophy and approach towards information modeling. This new approach includes a process that is founded upon software engineering techniques.

2. Information Overload

Three factors lend themselves to the information overload phenomenon: (i) the focus and expertise of the individuals who create the models; (ii) the level of data abstraction used in the models; and (iii) the desire to differentiate management and foster NIH ("Not Invented Here" syndrome).

2.1. Focus and Expertise

For organizations developing products such as network equipment, servers, and storage devices, management-software development is typically secondary to product-feature development. Consequently, more junior engineers are assigned to implement the management of those features, and often concentrate only on one particular aspect of management, such as configuration or fault management, versus a more complete management story. They are usually not familiar with the technical details of the product features, and focus on the idiosyncrasies of the languages used to express Management Information Bases (MIBs [4]), Policy Information Bases (PIBs [5]), and Common Information Model (CIM) Schemas [6]. Thus, engineers that specialize in management often have different knowledge, focus and skills than the feature developers.

Regarding schedule, management often lags behind product development—at least until the product features and management interfaces are defined. Customers require manageability, but they do not typically dictate normalized, standard semantics. The quantity and quality of data supplied is also not specified. Therefore, when schedules slip (as they customarily do!), management support can be compromised to fit the schedule.

Under these circumstances, supplying a MIB becomes a "check mark" item. Supplying standard semantics is often not a requirement. This allows a company to provide proprietary, non-normalized data in a MIB, using its own attribute names and terminology. This approach meets the requirement to supply management software. Unfortunately, customers or tool vendors are now required to mediate the terminology and concepts into a normalized form, or to create specific tools for each MIB.

2.2. Level of Data Abstraction

Finding the right level of abstraction is a difficult problem. The "right level" depends on how the data will be used. Some applications require high-level, very abstract data, while other applications (such as root cause analysis) may need detailed information on specific components of the management domain. Providing both levels of information as well as relationships between them can solve the dilemma. Using an object-oriented design approach and associations between classes (and therefore levels of abstractions) is one mechanism.

Pertinent to the discussion of levels of abstraction is a change of focus regarding how management information is defined. Often, management data is published simply because it is there. In this scenario, the rationale for publishing management data is trivial: Publishing something is better than publishing nothing at all, or publishing what is difficult to determine.

We claim that management requires design, and complexity can often be dealt with by defining use cases. Determining the problems to be solved, the data needed to solve those problems, and how the data should be traversed and utilized are important decisions. Object-oriented techniques and UML diagrams have a track record of helping solve these problems.

There are other issues with data abstraction. Abstraction and object-oriented design are difficult. Many standards committees do not want to standardize on the data, but on the mechanism for delivering it. One example is the Internet Engineering Task Force's (IETF) Simple Network Management Protocol (SNMP): The specific bits of a protocol are much more concrete than the more esoteric data needed for management. It should be noted that although the IETF has already defined more than 100 standard MIBs they are too often limited in the scope of products that they cover and rarely reuse concepts and constructs.

Lastly, data abstractions and management models must be extensible. This allows extension of the standards as well as a means for vendors to provide additional information critical to their products. Having extensibility as a primary goal (as opposed to just defining specific data bits) removes the need to include detailed information that only applies to a limited number of products.

2.3. NIH, the "Not Invented Here" Syndrome

During discussions about the relevance of standardizing management information, a frequent claim is that "I cannot use a standard management model since *my* environment (or *my* data) is unique." This is a difficult argument to refute without drilling down into the details of the environment in question. Most of the time, the argument is emotional and based on the perception that "my work goes where no man has gone before." A simple rebuttal is to ask what is so unique. For example:

- If modeling hardware, does no other product have a chassis or fit into a rack?
- If modeling software, does no other product have a version number and support specific functionality?

- If modeling routing, does no other system have routing tables and a concept of next-hop addressing?
- If modeling storage, does no other product have a volume?
- If modeling security, does no other product have credentials?

The obvious answer to all these questions is “of course they do”. So, with some effort, commonality can be found, abstractions can be defined, and standard semantics can be used and (more importantly) reused.

3. Common Information

With this new mindset, the goal becomes the creation of a single information model[1], defining common semantics and supporting various levels of abstraction for a given management domain. The translation of the model into a specific protocol and encoding renders a data model. The information model must be able to represent different views and abstractions of the management environment—for example, describing the configuration or operation of Quality of Service (QoS) processing behind a router, or indicating a network topology. The data from various products and vendors must be extensible, yet fit together to create a complete description of the environment.

In order to achieve this in a standard manner, the existing standards organizations (e.g. IETF, DMTF) need to adopt an efficient process to develop valid and sufficiently abstracted models. In addition, the standards bodies must work together to reuse modeling abstractions and constructs, and to avoid creating new terminology. In short, the bodies must work in conjunction rather than compete with each other.

3.1. Multi-tier Development Model

We propose that information modeling be developed using a multi-tier development model (see Figure 1REFSTYLESEQARABE). The top tier of the model is the high-level abstraction layer for a technology or management domain[2]. The Abstract Technology Model captures the main concepts of the domain and the relationships between these concepts. At this stage of development, it is important to ignore low-level data details, and focus on the concepts and goals of the management model. Work on the top tier should be done first by the technology and domain experts. Then, the model should be normalized and further designed by developers who are experienced in object-oriented technologies and familiar with the concepts of the Common Abstraction Model (see Section 3.1.2). (Note that there is only a single “Common” Model. This enables the technology and management domains work to fit together and be reused.)

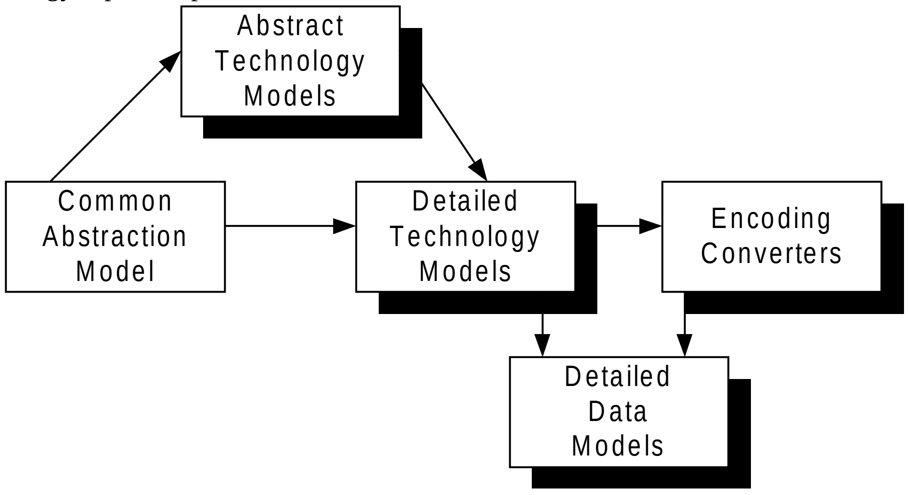
Once the Abstract Technology Model is complete, work begins on adding low-level data details to it and fleshing out the contents, objects and associations. At this stage of development, the model may go through several iterations of adding detail (based on input from the technology experts), conducting reviews and analyses

[1] See Martin-Flatin’s concept of Universal Information Model (UIM) [1, 7].

[2] In the figure, shadowed boxes imply that multiple models exist, for example, for different technologies.

against use cases, and normalizing the design using the Common Abstraction Model (by the developers of the common model). The result of this work is the Detailed Technology Model.

The Common Abstraction Model provides the basic concepts and root for all the technology models. It is built by promoting general concepts of the technology models (i.e., the concepts applicable to more than just one technology domain). The Common Abstraction Model developers are responsible for working with the technology experts to promote the correct classes and associations.



REFSTYLESEQARABEFigure 1: The Multi-Tier Development Model

Lastly, after the Detailed Technology Model is finalized, it can be converted to a variety of encodings. Some examples are SMI for use with SNMP [9], MOF for use with CIM [6], and/or LDIF for use as a directory schema [10]. The result of this encoding is a Detailed Data Model.

3.1.1. Abstract Technology Models

The top tier of the model gives the "big picture" for a technology. There may be models for routers, Storage Area Networks (SANs), relational databases, etc. There are as many Abstract Technology Models as there are teams that wish to provide standard data for managing these domains. With our approach, the increase in the quality of these models comes from the multi-tier methodology for designing these models, and from the reuse of common abstractions and the structured object hierarchy of the Common Abstraction Model.

The group of individuals working on the Abstract Technology Model should be a mixture of individuals:

- Technology experts (with detailed knowledge of the technology),
- Customers (defining the management problems and use cases), and
- Information modeling experts (providing expertise on object-oriented design and the Common Abstraction Model).

3.1.2. Common Abstraction Model

The high-level classes of the DMTF's CIM Core Schema [6] describe general concepts useful in a Common Abstraction Model. Following our philosophy of reuse and avoiding duplication of effort, we recommend the adoption of this model as a start toward normalizing data for all technology domains.

There are several words of caution, however. It is essential to avoid overloading the basic abstractions with detailed data (as has happened to some of the DMTF classes). A minimalist approach should be taken. Data should be included that meets the 80/20 rule (applicable to 80% of all users and subclasses of the model). Also, we avoid adopting all the language constructs of the DMTF in this discussion, and focus solely on concepts.

The purpose of the Common Abstraction Model in Figure 1REFSTYLESEQARABE (and the CIM Core Schema) is to introduce basic concepts that bring organization and common semantics to objects in the Abstract Technology Models. Some of the abstractions recommended from the CIM Core Schema are: ManagedElement, Collection, Setting, StatisticalInformation, PhysicalElement, LogicalElement, LogicalDevice, System, Service, and ServiceAccessPoint (seeRENV Figure 2). Also depicted in this figure is the Capabilities class. This class is proposed for inclusion in the CIM Core Schema by several of the authors. The common base associations also recommended are: Dependency, Component, AssociatedStatistics, ElementSetting, Element Capabilities and MemberOfCollection. Proposed by the authors is a new association, "SeeAlso".

Although the class names are fairly intuitive, some overview of the concepts and their relationships is required for completeness. ManagedElement is the root for all non-association class definitions and defines the key structure and basic properties (such as Description) of the object hierarchy. Dependency describes both directional and peer dependencies between two instances. Component indicates a whole-part (composition) and/or containment relationship between instances. (Although not specifically noted in Figure 2, associations are full classes and can define properties beyond simply references.) Lastly, SeeAlso relates different aspects of a real world element, instantiated through different classes of the object hierarchy. An example is a keyboard that is also a USB device. This could be represented using multiple inheritance – deriving from both the KeyboardDevice and a USBDevice class. Unfortunately, many implementations do not support multiple inheritance. Hence, an association that describes the basic concept is provided – and implementations are free to use whatever mechanisms are available to manipulate or realize the association.

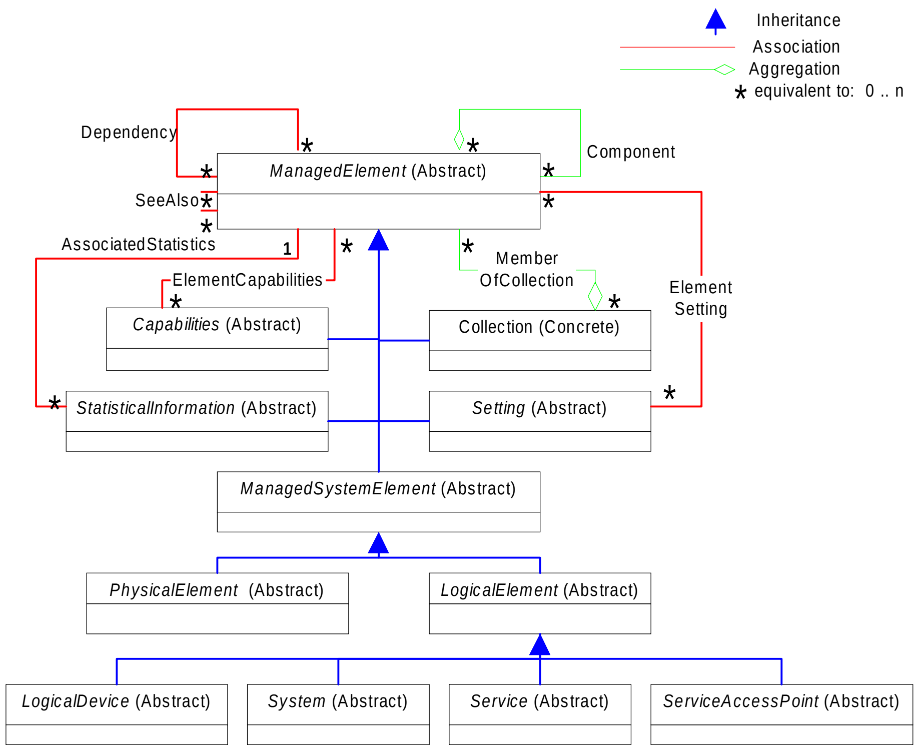
It may be asked why classes such as Capabilities, Settings or Statistical Information are defined. The base classes themselves could be defined to contain the properties found in these other classes. However, some capabilities, statistics, and setting information are dependent upon the usage of the instance versus the class itself. Therefore, having separate classes to describe these properties is more flexible than burdening the class definition with all possibilities. Also, depending on the users' eventing and notification schemes, it may be more straightforward to notify on any change to a base class – and less often on changes to statistical data. The associations ElementCapabilities, AssociatedStatistics, and ElementsSetting relate the Capabilities, Statistical Information, and Settings values to the object(s) to which they apply or that make use of them.

Collection is a bit strange as a fundamental object. Programmers understand its use in a class library, but do not often think of it as a management concept. It is useful, however, as a way to collect instances into a "bag". An example is a group of Settings that apply to a particular user. This could be defined as a subclass of Collection (a Profile) that gathers together the appropriate instances of (subclasses of) Settings; the MemberOfCollection association is used to express the instances that are "collected". Collection is different from the Component association since it does not describe a composition or containment relationship but only basic aggregation.

So far, the instances of classes that we described (such as StatisticalInformation) would not have statuses. Therefore, ManagedSystemElement is intended as the parent to all classes that are parts of "systems" (a very general concept) and that have status. Status defines the basic operational states of an element using one standard set of terms (OK, Error, Failure Predicted, Non-Recoverable Error, etc.). This concept aligns with that of M.3100 OperationalState [3]. No longer is it necessary to translate all synonyms of "OK" ("good", "functional", "working", etc.) to determine status. It should be noted that subclasses can expand on these basic states using new properties, specific to their problem domain. For example, a printer class may include a property, ErrorInformation, that details specific "Error" states (such as low toner and paper jam). (In addition, the Common Model provides other properties, that are not discussed here, to describe concepts such as enabled/disabled state.)

It is desirable to have a clean separation between describing the physical world and the logical world. Often, different mechanisms or software are used to obtain this information. Therefore, PhysicalElement is the parent for the class hierarchy that describes the physical world (things that must adhere to the laws of physics and that can be seen or touched). LogicalElement is the parent of the class hierarchy that describes things in a logical sense. Under LogicalElement is where most of the management occurs. PhysicalElements can be moved, replaced, powered, etc. but not much more. The subclasses of LogicalElement are:

- **LogicalDevice -** Describing the specific functionality provided by the physical hardware (such as a KeyboardDevice)
- **System -** Describing entities that are composed of ManagedSystemElements and which are managed at a high level as a single entity (such as a NetworkElement or an AdminDomain)
- **Service -** Describing any functionality, especially that provided by the "running" of software (such as RoutingService)
- **ServiceAccessPoint -** Describing the means of accessing Services (such as ProtocolEndpoints that access network Services)



REFSTYLESEQARABE Figure 2: High-Level Common Abstraction Objects

The authors acknowledge that agreeing on common interpretations, terminology and abstractions is very difficult. However, it is not impossible as proven by the work of the DMTF.

3.1.3. Detailed Technology and Data Models

The most specific tiers of the management models are those of the Detailed Technology and Data Models. The Detailed Technology Models define the low-level details of the technology or management domains, addressing the needs of the use cases. The Data Models are the transformation of the Technology Models into specific syntaxes, for transmission using specific protocols, possibly destined for specific repositories.

It is required that the terminology used to express the Detailed Data Models match that of the Detailed Technology Models, to avoid confusion and divergence. However, it is also anticipated that some changes will be required as the models are adapted to specific syntaxes, and the idiosyncrasies of those syntaxes are taken into account. It is desirable for both the Technology and Data Models to be standardized. For example, for IP-based models, MIBs and PIBs would be published by the IETF, while the DMTF would produce MOF files. Either standards group (or ideally both in conjunction) would publish the Detailed Technology Models.

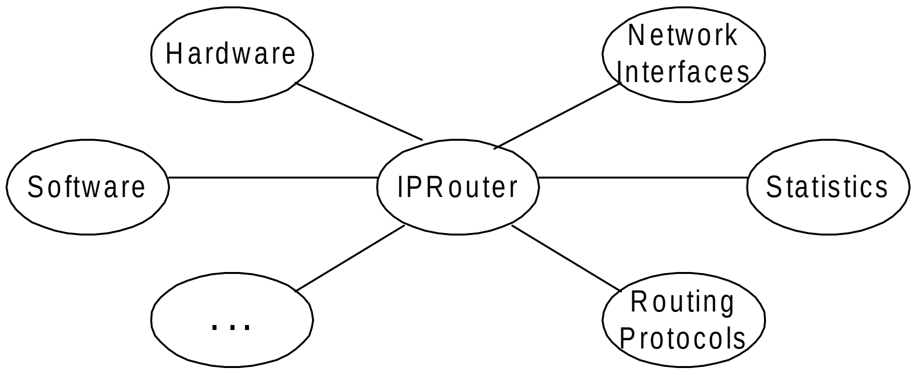
4. Example

In order to make the preceding discussion more understandable, a concrete example is helpful. For the example, our technology model is an IP Router. We map, normalize, and expand data from the IETF's RFC 1213 [8] and include mapping of information for the ITU's M.3100 specification [3]. A lighter-weight version of the DMTF CIM Core Model is used as our Common Abstraction Model (This model was briefly overviewed in 3.1.2)[3]. In addition, various concepts from DMTF's CIM Network Model are also discussed as part of our Common Abstraction Model.

Since this paper cannot address all aspects of a router, RFC1213 or M.3100 (due to limited space), the authors request some leeway regarding the completeness of this analysis. The focus is on the use of the development model presented in Section 3, as it pertains to a specific technology domain.

Partitioning the management of an IP Router into its logical aspects such as interfaces, statistics, hardware, routes and routing protocols is a necessary first step in creating an Abstract Technology Model. The managed information is partitioned as shown inRENV Figure 3.

Once the "abstract" concepts of the technology model are identified, the next step is to map the management information into the Common Abstraction Model. The IP Router maps to the abstract model as an instance of the ManagedNetworkElement class whose inherited property, Dedicated, is used to indicate that the system is dedicated to routing (see RENVFigure 4 where the model and its properties are explicitly shown).

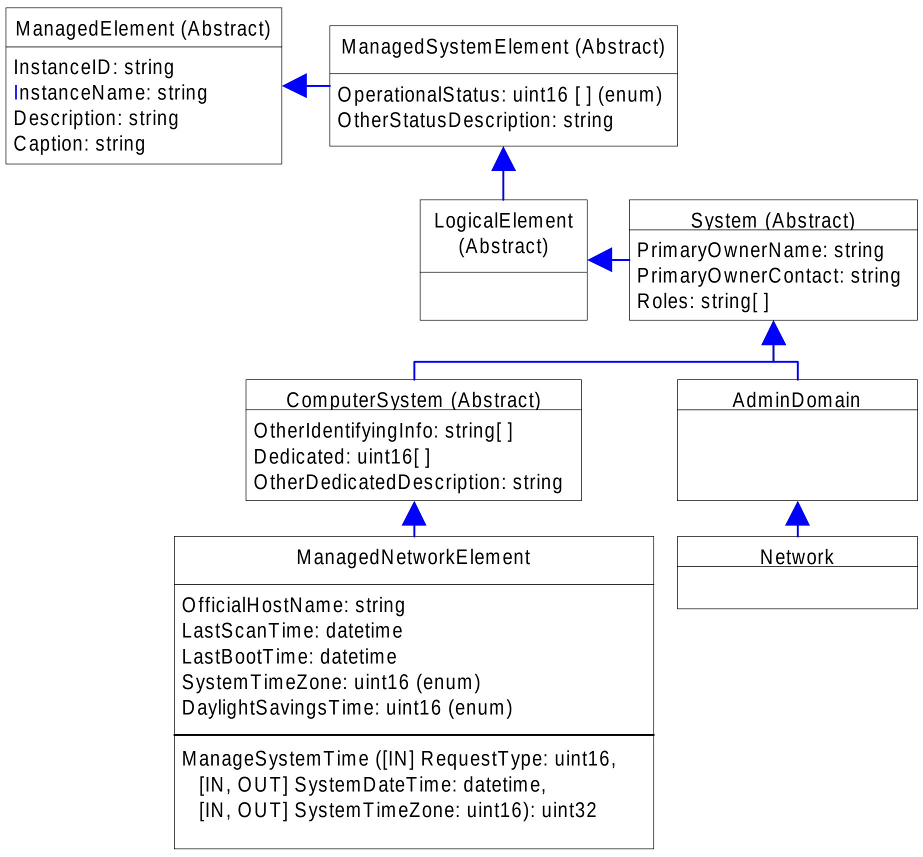


REFSTYLESEQARABEFigure 3: IP Router Functionality Technology Model

It is also important to describe how the IP Router is managed in the context of a Network. As can be seen in the Figure 4, an instance of a Network (corresponding to ITU's M.3100 Network) is a subclass of AdminDomain. Reviewing the object hierarchy, it is seen that both the IP Router and its Network are instances of a System (a very general and abstract concept). They are managed as single entities that are

[3] The authors have chosen a "lighter weight" version of the DMTF CIM Core Model to simplify the discussion.

the sum of their components (i.e., their hardware, software, services, other systems, etc.).


ManagedElement (Abstract)
InstanceID: string
InstanceName: string
Description: string
Caption: string
ManagedSystemElement (Abstract)
OperationalStatus: uint16 [ ] (enum)
OtherStatusDescription: string
LogicalElement (Abstract)
System (Abstract)
PrimaryOwnerName: string
PrimaryOwnerContact: string
Roles: string[ ]
ComputerSystem (Abstract)
OtherIdentifyingInfo: string[ ]
Dedicated: uint16[ ]
OtherDedicatedDescription: string
AdminDomain
ManagedNetworkElement
OfficialHostName: string
LastScanTime: datetime
LastBootTime: datetime
SystemTimeZone: uint16 (enum)
DaylightSavingsTime: uint16 (enum)
ManageSystemTime ([IN] RequestType: uint16, [IN, OUT] SystemDateTime: datetime, [IN, OUT] SystemTimeZone: uint16): uint32
Network

REFSTYLESEQARABEFigure 4: IP Router Common Abstraction Model

When adding detailed data to the general concepts of the Common Abstraction Model and Abstract Technology Model, the authors looked to other standards bodies for insight. In particular, the detailed data of RFC 1213 and the ITU M.3100 specification were considered. RENVTable 1 shows the relationship between the properties of the router’s Detailed Technology Model and the MIB attributes from the IETF RFC 1213.

In the next phase of model development, it is important to drill down beyond the systems themselves to more detailed aspects of the IP Router, such as its interfaces, routes and routing services. These aspects are not part of the System class, but individual concepts associated with the System (router).

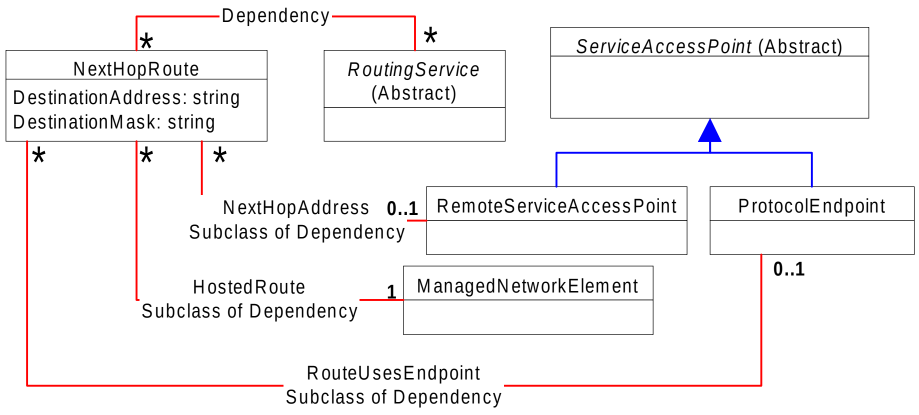
REFSTYLESEQARABETable 1: Model Properties and 1213 MIB Attributes

Abstract Technology Model Property Name	RFC 1213 MIB Attribute Name
InstanceID	sysObjectID
InstanceName	
Description	sysDescr
Caption	
OperationalStatus	
OtherOperationalDescription	
PrimaryOwnerName	sysContact
PrimaryOwnerContact	sysContact
Roles	sysServices
OtherIdentifyingInfo	
Dedicated	
OtherDedicatedDescription	
OfficialHostName	sysName
LastScanTime	
LastBootTime	sysUpTime
SystemTimeZone	
DaylightSavingsTime	
Location	sysLocation
(part of the PhysicalElement hierarchy)	

Network interfaces are subclasses of ServiceAccessPoints, hosted by the IP Router, permitting access of the network, other systems and the local protocol stack, and used to describe the layering of the protocols in the stack. They are equivalent to the concept of "termination points" defined in ITU's M.3100. Network interfaces map to the abstract model as subclasses of ServiceAccessPoints, RemoteServiceAccessPoints and ProtocolEndpoints. Examples of ProtocolEndpoints are IPProtocolEndpoint, LANEndpoint, UDPProtocolEndpoint, etc. RemoteAccessPoints hold the protocol information used by one system to contact a "remote" system.

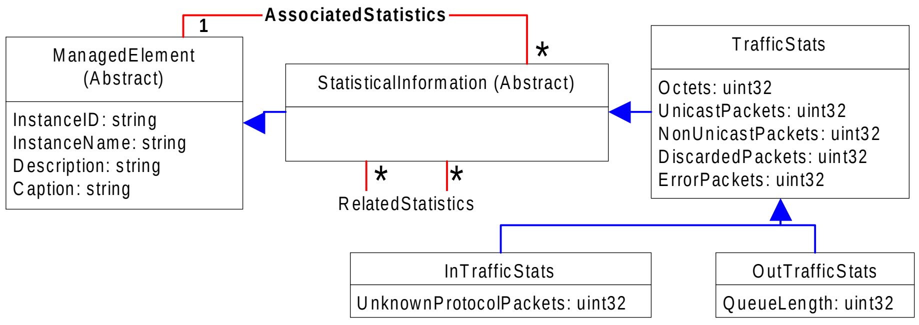
A ServiceAccessPoint has several specialized relationships. It uses an association to indicate the system that hosts it. (This association can be directly translated to containment in implementations that support this concept.) It can have Dependency relationships to entities that use it. A subclass of Dependency association (BindsTo) shows the layering of the Endpoints in the protocol stack.

The concepts of RoutingServices, ProtocolEndpoints, and NextHopRoutes are related as shown inRENV Figure 5. A RoutingService is Dependent on the use of static and dynamic routes to reach a target (destination) address. These NextHopRoutes are typically defined within the context of the Router or Switch, as described by the HostedRoute association. In addition, the NextHopRoute describes the mechanism of getting to the next "hop" via a specific Endpoint and/or RemoteServiceAccessPoint. The particular instances used for a route are described using the associations, RouteUsesEndpoint and/or NextHopAddresss. Both of these associations are optional.



REFSTYLESEQARABEFigure 5: Routing and Routing Protocols Example of the Abstraction Model

A benefit of the iterative approach and object-oriented techniques is small incremental cost to defining additional classes. In addition, there is advantage to not burdening all instances with data that may not be available or applicable. This applies well to statistics and allows the statistical information contained in the Interface table of RFC 1213 to be logically grouped into specific classes, and not forced into the general ProtocolEndpoint class. ProtocolEndpoints can be associated with the statistical information that is applicable to them. Therefore, not every ProtocolEndpoint is burdened with all statistical properties. For example, RENVFigure 6 shows a set of statistics classes that are related to traffic (based on attributes in RFC 1213).



REFSTYLESEQARABE Figure 6: Statistics Example of the Abstraction Model

Although many more details of the IP Router could be discussed, this is left to another paper. However, it is important to consider one additional item with respect to the Models – that of implementation. Once the Detailed Technology Models are sufficiently complete, they are translated into various syntaxes for implementation. For example, the Models can be translated into one or more IETF MIB(s), DMTF

In *Proc. of the Network Operations and Management Symposium (NOMS 2002)*, Florence, Italy, April 2002.

MOF(s) and/or used in APIs. This permits instrumentation to work in the "standard" protocols, while delivering information that is "mediated and normalized." Implementation is the realization of the abstractions and concepts from the Detailed Technology Model and Common Abstraction Model.

5. Related Work

Why was CIM chosen, and not one of the many other standards? One reason is the desire of the DMTF to address the complete management space (systems, networks, telecommunications, databases, users, policy, and more). Another reason is to map other standards into the CIM Schemas. A third reason is the very powerful, high-level abstractions of the CIM Schemas (especially the Core Model) adequately model the wide variety of concepts needed by the authors.

Admittedly, many other standards could have been chosen – for example, the M.3100 standard from ITU. Its drawback, however, is that it is not typically used outside of the telecommunications environment. On the other hand, SNMP and MIBs from the IETF are widely deployed in network devices. The IETF Working Groups have never really focused on defining common abstractions and reusable concepts. Unfortunately, standard access does not translate to standard data. SNMP MIB(s) are typically focused on the Network Element Layer and are not used for provisioning and configuration, nor is SNMP implemented in all problem domains.

Combining the best of all worlds, CIM can take M.3100 and other ITU standards, as well as standard SNMP MIB data (for example, RFC1213), and organize the information using the CIM hierarchy. The other standards are not ignored or dismissed. They can be mapped into the CIM object model.

However, no standard is perfect and CIM is still rather new in the standards space (having been defined in the mid 1990's). Several problems in CIM were encountered in areas such as keys/naming, model consistency and model complexity/heaviness. These are being addressed as part of future CIM releases.

6. Conclusion

In the first part of this paper, we have presented a multi-tier model that solves the issue of non-normalized management information overload and improves the design of information models. For each technology, we define three models through an iterative process: an Abstract Technology Model, a Detailed Technology Model and a Detailed Data Model. The multiple iterations ensure the quality and stability of the data models eventually used by the vendors and customers.

In the second part, we have studied the example of the management of an IP Router, and built upon the work already performed by the IETF (RFC 1213) and DMTF (CIM Core and Network Schemas) to define some of the technology-specific models required to manage an IP router. It is the intent of the authors to further expand this mapping and example in other papers and presentations.

In the future, we intend to refine the Common Abstraction Model while defining new Abstract Technology Models. Another interesting direction for future work is to improve the models for managing an IP router, in order to better highlight the commonalities and discrepancies between the ITU, IETF and DMTF efforts. Work is needed to analyze the cost benefits of a single information model versus a federated approach.

References

[1] J.P. Martin-Flatin, "Toward Universal Information Models in Enterprise Management", in W. Jonker (Ed.), *Databases in Telecommunications II—Proc. VLDB 2001 Workshop on Databases in Telecommunications (DBTel 2001)*, Rome, Italy, September 2001, LNCS 2209:167–178, Springer, 2001.

[2] A. Westerinen and J. Strassner, *CIM Core Model Whitepaper*, DMTF DSP0111, August 2000.

[3] ITU-T, M.3100, *Generic Network Information Model*, 1995.

[4] K. McCloghrie and M. Rose (Eds.), RFC 1212, *Concise MIB Definitions*, IETF, March 1991.

[5] M. Fine, K. McCloghrie, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, and F. Reichmeyer (Eds.), *Framework Policy Information Base*, Internet-Draft <draft-ietf-rap-frameworkpib-06.txt> (work in progress), IETF, November 2001. Expires May 2002.

[6] DMTF, *Common Information Model (CIM) Specification, Version 2.2*, June 1999. Available at <http://www.dmtf.org/standards/cim_spec_v22/>.

[7] J.P. Martin-Flatin, D. Srivastava, and A. Westerinen, *Iterative, Multi-Tier Management Information Modeling*, Technical Report TD-4XURR2, AT&T Labs Research, June 2001.

[8] K. McCloghrie and M. Rose (Eds.), RFC 1213, *Management Information Base for Network Management of TCP/IP-based internets: MIB-II*, IETF, March 1991.

[9] K. McCloghrie, D. Perkins, and J. Schoenwaelder (Eds.), RFC 2578, *Structure of Management Information Version 2 (SMIv2)*, IETF, April 1999.

[10] G. Good (Ed.), RFC 2849, *The LDAP Data Interchange Format (LDIF) - Technical Specification*, IETF, June 2000.

Biographies

J. Schott is a Technical Leader for the Information Modeling group at Cisco Systems. She has worked in the industry for over 17 years in the areas of network and storage management and software development. Julie is the current chair of the DMTF CIM System and Devices (SysDev) Working Group and is a participant in the Storage Networking Industry Association (SNIA). Before joining Cisco, Julie was employed by LSI Logic, Maxtor, Intellistor, Information Storage, and Hewlett-Packard. She has a B.S. in Computer Science from Iowa State University.

A. Westerinen is a Senior Architect and Manager of Information Modeling at Cisco Systems. She has worked in the computer industry for more than 20 years, the last eight years principally in the areas of enterprise, system, network, storage, and policy-based management. Andrea is the Vice President of Technology of the DMTF, and an active participant in the IETF and TM Forum. She is an expert on the CIM schemas published by the DMTF, and is currently acting chair of the DMTF CIM Networks Working Group. Andrea has co-authored a book, *Common*

Information Model, as well as several IETF documents on policy-based management. Before joining Cisco, Andrea was employed by Microsoft, Intel, IBM, and NCR. She has a B.S. in Physics and Mathematics from Marquette University, and an M.S. in Computer Science from Nova University.

J.P. Martin-Flatin is a Principal Technical Staff Member with AT&T Labs Research. He holds a Ph.D. degree in Computer Science from the Swiss Fed. Inst. of Technology, Lausanne (EPFL). He has worked 10 years in industry in the areas of network and systems management, software development, security, and Web engineering. He is the author of a book, *Web-Based Management of IP Networks and Systems*, and was a guest editor for a special issue of the Journal of Network and Systems Management on the same topic. The bottom line of his research is to build an interdisciplinary bridge between enterprise management and software engineering. He is a member of the IEEE, ACM, IRTF Network Management Research Group, and the DMTF SysDev and Interop Working Groups.

P. Rivera is a Principal Engineer with LSI Logic. He has worked in the computer industry for over 18 years in the areas of storage management and software development. Peter is an active participant in the CIM SysDev Working Group of the DMTF. Before joining LSI Logic, Peter was employed by Sun Microsystems and Digital Equipment Corporation. He has a B.S in Computer Science and Electrical Engineering from the University of Colorado at Boulder.