Mapping Management Information to Service Interfaces Supporting High-Level Network Management Applications

Michael Stadler
OFFIS Research Institute
Escherweg 2
26121 Oldenburg, Germany

e-mail: stadler@offis.uni-oldenburg.de

Abstract

Management information available to network management applications is mostly technically detailed. In addition, high-level management applications need more abstract models of networks and their components. In order to automate provision of abstract models at management service interfaces, it is necessary to define the mapping between abstract models and the underlying management information, provided by agents and management platforms. For this purpose we introduce GDMO-M, a language for mapping GDMO-defined managed object classes to basic management information which may be structured in different ways.

1 Introduction

Despite the continuing efforts for automating network management, there are still few management applications, that really contribute to automation. While today's management applications may take the role of helping hands and binoculars, the thinking has still to be done by network operators themselves, even for routine tasks. There are some exceptions to this statement, but the respective applications are monolithic pieces of software, which incorporate their own basic management infrastructure.

A main reason for this situation is the lack of support that today's management systems offer to their management applications. It is difficult to build high-level management applications based on low level functionality.

2 Management service interfaces

Management service interfaces allow management applications to access a basic infrastructure realised by agents, management platforms and other systems which offer access to management information. Common examples of such interfaces are CMIS or SNMP-interfaces offered by the agents of management systems, or the application programming interfaces offered by management platforms.

The primary aspect of a management service interface is its functionality. Main functions offered at every interface allow reading of management information, changes of management information and reception of notifications. However, management service interfaces differ in the options they offer to management applications for using these functions. E.g. some interfaces allow retrieval of historical information or they might allow an application to request a certain degree of actuality for the information delivered. Functionality is one aspect of management service interfaces which may be improved in order to give a better support to high-level management applications.

The usefulness of functionality concerning retrieval and change of management information depends upon the network models and component models which may be accessed through the interfaces[1]. Today's management systems offer device oriented models that exhibit a high-level of technical details.

Mostly, the models are supplied by the management system's agents. These models are described in terms of standards which define the structure of management information [IS 10165-2], [RFC 1442]. In few cases only, management platforms extend the models supplied by agents or provide supplementary models, i.e. for representing subnets. However, the information offered by these models is very limited and often they are not accessible through management service interfaces, but serve for internal purposes of management platforms or for graphical representation.

The level of abstraction of models offered by today's management systems is often not sufficient for supporting high-level management applications. For this reason, many applications have to build network models of their own. This introduces a considerable difficulty in application development. Things would be much easier, if abstract models (we use this term to describe models that are more abstract than those presented by today's agents and management platforms) could be offered to management applications by a supporting infrastructure. I.e. abstract models should be modeled in a uniform way and be accessible through management service interfaces.

[1] In this paper we use the term "model" to designate the representation of structure, configuration and state of a technical system to be managed.

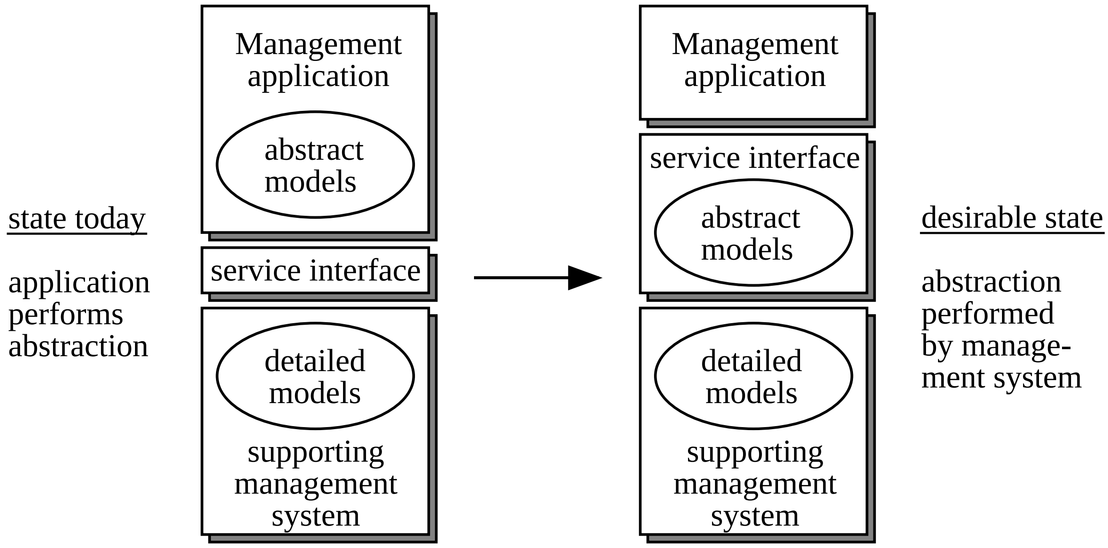

Fig. 1

3 Requirements on Models supporting High-Level Management Applications

While today's models mostly represent single devices, high-level management applications (HLMA) often need a view that includes multiple devices which are related by their function. Therefore, models representing entities that consist of multiple network components are needed. Besides attributes that give detailed technical information as e.g. the number of bytes sent through a network interface, HLMAs need more abstract information. The latter allows to tackle down problems in a top-down process. During a problem solving step abstract information may be used to guide the focus of interest. I.e. abstract information may give an orientation about where to continue problem solving and about what to get more detailed information.

Abstract information may be derived from other information by numerical or logical computation. As an example, we consider a model representing a subnetwork containing multiple communication links. The interface specific information of devices attached to a network, is the lowest level of information we take into account. From this, an attribute representing the current link usage may be derived for every link present in the subnetwork. From these attributes, in turn an attribute giving information about the current average link load in all the subnetwork may be computed. As a next step it would be possible to derive an attribute which may take the values "load ok", "local overload", "extensive overload". For computing these values, time series of the values the underlying attribute exhibit should be considered. As a last step an attribute could be defined, giving still more abstract information about the general subnetwork's state. This might adopt the values "no action required" if network operation is normal, "tuning required" if there are no abrupt transitions in network load and "possible fault detected" if there was a direct transition from "load ok" to "extensive overload" in the attribute defined previously. Figure 2 illustrates the growing degree of abstraction.

Level of abstraction

Attribute name	Attribute values	assigned to object type
administrative info	no action required, tuning required, possible fault	
load state	load ok, local overload, extensive overload	network
current average link load	numerical	
current link usage	numerical	link
#bytes transmitted per second	numerical	device

Fig. 2

To summarize, the two model characteristics that are often needed by HLMAs but not required by today's management applications are:

- Models must be able to represent network components which are composed of multiple active network devices (e.g. open systems)
- Qualitative domains must be allowed for the model's attributes

Abstract models are to be used in conjunction with today's device specific models, because HLMAs need those models, too. Therefore, the former should be accessible by the same methods as the latter. Furthermore, tools used in conjunction with model definition should be usable for abstract models as well as for device specific models.

4 The procedure for providing abstract models

In order to provide abstract models of networks and their components at management service interfaces, different steps have to be taken. One part of these steps has to be carried out in order to define the models. The other part is carried out for defining the mapping between the abstract models and the underlying management information. We will now give a more detailed description of the actions to be taken. Figure 3 gives an overview.

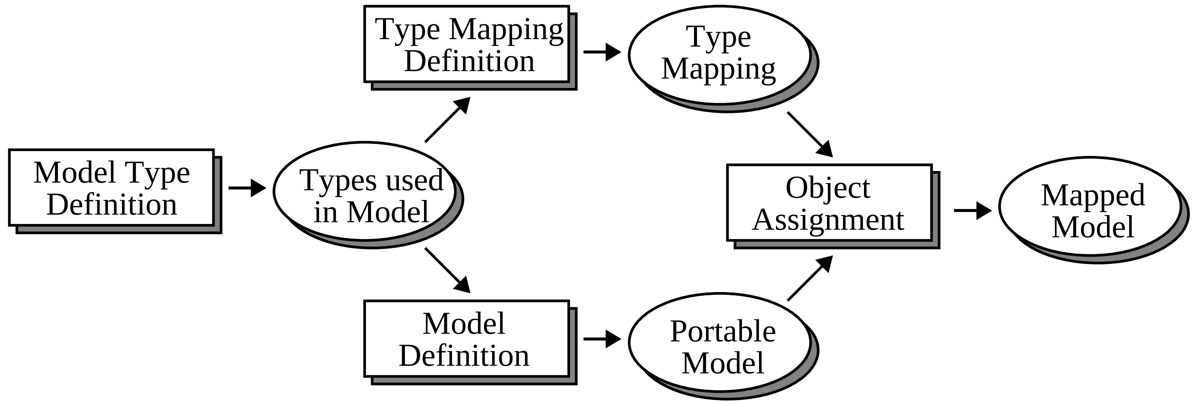

Fig. 3

4.1 Defining Types used in Models

The models under discussion are composed of objects and relationships. Thus, before designing a model, types of objects and relationships have to be defined. In this context the term objects refers both to attributes and to object instances. In the latter case object classes have to be defined and in the former case domains of variables. This is done in a process we call *model type definition*. In order to carry out this step, a language in which to express type definitions must be devised. Today, there are three well-known languages for supporting model type definitions. One is GDMO [IS 10165-4], which is used when defining models used in the context of the OSI management framework [IS 10040]. Another language is DML [DOMAINS 92], defined in the DOMAINS Project. And last but not least there is the set of ASN.1-Macros used to define object types in conjunction with the IAB management framework [RFC 1155].

The requirements on abstract models mentioned in section 3 induce requirements on the language used for model type definition, too. First, it should allow the definition of types representing network components consisting of multiple active devices. This is best accomplished if an object oriented language is chosen. Second, the definition of qualitative domains must be possible. And third, resulting type definitions should be compatible to existing model type definitions.

Based on these requirements we propose GDMO as language for defining types used in abstract network models. However, the detailed layout of the selection is out of the scope of this paper. To our knowledge, catalogues of GDMO-defined managed object classes available today, are based on the assumption, that no managed object may represent a system, which in turn contains multiple open systems. This is no limitation imposed by GDMO. So we feel free to assume that suitable class catalogues will serve as a basis for defining abstract models.

4.2 Defining Models

Given types of objects and relationships, models may be defined, consisting of object instances and relationships between these instances. This takes place during *model definition*.

In today’s management systems models at the device level are implemented by agents. To transport such a model from one management system to another, the agent implementing the model has to be ported. This is a common practice because usually agents are shipped with the devices to be managed. Models representing network components that consist of multiple devices, today only occur at the manager level, e.g. for graphical representation of subnets. Those models are specialized for use with one management tool at one location. Thus it is not necessary to port those models. For backup purposes dumps of management platform databases are sufficient.

There is no need for a model definition language, because database dumps are a sufficient means for backup and transporting models between different locations and tools is not necessary.

If models are introduced that abstract from technical details and represent composite network components, the situation changes. In this case the same model may well be used at different sites. Furthermore, since models are more complex, their definition requires more work than the definitions of device specific models. For this reason, transporting models becomes meaningful and a publicly available model definition language has to be devised to allow for portability.

4.3 Mapping Models

Before a model may be offered to management applications at a management system's interface, a mapping between the model and the available management information has to be defined and implemented. First, during the step of *type mapping definition*, a mapping between the object types present in the abstract model and the types present in the underlying management information has to be defined. Second, after porting a model to a particular site, an assignment of information sources to the object instances present in the model has to occur. This takes place during the *object assignment* step, which we will not deepen.

In the remainder of this paper, we will describe the concept of mapping between models and available management information in more detail. We will especially focus on a language for defining type mappings.

5 A Language for defining Type Mappings

Our goal is to automate the mapping between abstract models and the underlying management information. This would ease the process of porting abstract models between different sites. Furthermore the implementation of new models becomes easier, if only the model definition has to be done manually and the mapping is carried out automatically as far as possible. One prerequisite for automatic generation of mapping between abstract models and underlying management information is a language in which to define type mappings. In Section 4.1 we proposed to use GDMO for defininig the types to use in abstract models. Thus, a language has to be defined in which the mapping between GDMO classes on one side and underlying management information on the other side may be defined.

We assume, that the underlying management information exists in different formats. Information may be given as objects with attributes and defined by GDMO or other object oriented languages. They might as well be given as structured collections of named variables as specified for IAB-

MIBs [RFC 1155]. Most operating systems or application programs present their information as more or less structured sets of named variables, too. Note that the underlying management information may as well be looked upon as one or multiple models of network components.

Currently, there exists no entirely specified language for defining model type mappings. However, in [Neumair 93] the concept of „Mapping Managed Object Classes“ (MMOC) is introduced. MMOCs are not classes, but if a mapping should be defined for a managed object class (MOC), this is done by adjoining a MMOC to it. An MMOC contains a mapping rule for each of the attributes of its adjoined MOC, but syntax and semantics of those rules are either not published or have not been defined.

Keeping the idea of adjoining a construct to every GDMO-defined MOC, we developed the language GDMO-M. We will present this type mapping language after presenting the requirements it has to fulfill.

5.1 Requirements

There are several requirements, that have to be fulfilled by a language for defining the mapping between abstract models realised with GDMO-defined objects and underlying management information. One requirement, that the reader might think is missing, concerns the mapping of relationship types. We assume that each model has its own relationship structure which is defined separately for each model. Apart from this topic, the requirements we identified are:

- A suitable language must include a naming scheme for referencing management information from different sources. The addressing schemes for referencing management information differ between different providers of management information. Already the information offered by systems following different architectural approaches to management systems is different. Thus, to allow for reusability of type mapping definitions in different environments, a unified naming scheme has to be included.
- A type mapping language must support object selection by object properties. Object assignment can not be automatted entirely, while type mapping can. Thus, it is desirable to shift as much mapping information from object assignment to type mapping as possible. By allowing object selection through properties, under certain circumstances it is possible to identify objects contained in management information without having to name them. At least it is possible to reduce address information needed to reference objects. This eliminates the need of unique assignment from objects to names in the assignment step. Instead, sets of objects may be assigned, which alleviates the task of object assignment.
- It must be possible to map multiple elements of underlying management information to a single attribute of the abstract model. This can be of use, for example when an attribute must be computed from information, originating from different managed objects or even different sources of management information.
- The language must support mapping of multiple values to set-valued attributes of objects contained in the abstract model.
- Mapping of historic management information to attributes of abstract models must be supported.

- A suitable language must allow mapping between attributes with qualitative domains that are part of the abstract model, and underlying management information of arbitrary domains. High-level management applications often rely on qualitative management information. On the other hand, with growing level of detail management information is often given as numerical vaulues. This gap has to be bridged.
- It must be possible to map time relationships between management informations to attributes having a qualitative domain. E.g. it must be possible to express that one attribute of underlying management information had a given value before another attribute had another given value. It is desirable to represent such statements in abstract models by defining appropriate attributes with qualitative domains. The appropriate mapping of historical data to qualitative attributes must be defineable by means of a type mapping language.

Note that any type mapping language meeting the requirements above, may be used to define the mapping between objects within an abstract model as well.

5.2 Syntax and Semantics of GDMO-M

GDMO-M consists of a set of templates. A type mapping definition consists of multiple class mapping definitions, which are defined by MAPPING DEFINITION templates. In this template a MOC to be mapped is identified, together with a number of types of underlying management information to be mapped. For each package present in the GDMO-definition of the MOC to be mapped, the MAPPING DEFINITION template references a mapping package. Mapping packages are defined by the MAPPING PACKAGE template. Each mapping package definition references a set of mapping rule definitions, one for each attribute contained in the package to be mapped. Mapping rules are defined by the MAPPING RULE template.

```
ruleDefinition ::=
<ruleIdentifier>      MAPPING RULE
                      FOR ATTRIBUTE <AttributeIdentifier>;
                      ( VALUE IS <expression>
                      | <constantIdentifier> IF <logExpr>
                      (, <constantIdentifier> IF <logExpr>)*
                      [, <constantIdentifier> ELSE]);
```

The attribute to map is specified by means of the FOR ATTRIBUTE clause. After this, a rule for assigning a value to this attribute is given by the VALUE IS clause. Attributes having numerical domains may be assigned a value by giving a numerical or logical expression. The other option is reserved for attributes having qualitative domains (i.e. of ASN.1-type ENUMERATED). The IF clause serves the purpose of assigning a logical expression to a constant of the qualitative domain. This logical expression serves as characteristic predicate to the constant. If the expression is evaluated to TRUE, the mapped attribute is assigned the corresponding constant. The ELSE clause has been introduced for preventing undefined attributes.

Referencing Management Information in GDMO-M

One requirement to type mapping languages is the introduction of a scheme allowing to reference management information from different sources in the same way, even if they use different naming schemes.

We assume that referencing management information is a two-phase process. In a first phase types are identified and thereby the set of instances belonging to these types. An instance in this context may either be a classe's instance (i.e. an object) or a simple type's instance (i.e. a value). During a subsequent phase instances are selected by their properties. One example of such a two-phase process is that of scoping and filtering introced in context of OSI-management [IS 9595].

The two-phase process is supported by GDMO-M through two different concepts. The concept of naming supports the identification of types by their names and their organization. Furthermore it may allow preselection of object instances by their naming or the naming of information providers they are administered by. Identification of instances by their properties is supported by the concept of selection. We will present these concepts in the following sections.

The naming concept

GDMO-M applies a naming scheme which assumes that each type name may consist of a module identifier, a type identifier and an attribute identifier. A type name identifies all instances of types referenced by it. To more precisely identify instances an agent identifier and an instance identifier may be added to a type name, resulting in a smaller set of referenced instances. It is possible to omit components of a name, if they are not necessary to reference the desired sets of instances.

	Type naming			Type restriction	
	Module Identifier	Type Identifier	Attribute Identifier	Agent Identifier	Instance Identifier
ISO/OSI	OSI standard name or catalog name	GDMO class came	GDMO attribute name	Agent address	distinguished name
IAB	IAB MIB-Module name	Name of table entry mana-ged obj.*	Name of managed object	Agent address	row index* or zero
BSD UNIX	/dev/kmem	Struct name*	Variable Name	Host Address	array index*

* means: does not apply for scalar values and unstructured objects

Fig. 4: Mapping between GDMO-M naming scheme and other naming schemes

A name which consists of a module identifier followed by a colon identifies a module, that contains a number of type definitions. E.g. `IAB-MIB2:` might be used to identify MIB2 defined by

IAB in [RFC 1158]. The names used for identifying type modules – just as any other component names used for addressing – have to be defined during object assignment.

Structured types as records and classes may be addressed by a type identifier followed by a period. To avoid ambiguities, a module identifier may be given, too. E.g. the name `system.` could identify a class or structure assigned the name "system". The name `ArbitraryPlatform:genericSNMPDevice.` would uniquely identify a class or structure called "genericSNMPDevice" in a module named "ArbitraryPlatform" which could be the class library of a management platform.

Any simple type may be identified by giving an attribute identifier. E.g. `ifOutOctets` could be used to identify the corresponding type defined in MIB2. Ambiguities may be resolved by completing the name through concatenation with type and module identifiers.

The address types mentioned so far identify a set of types. E.g. the name `OSI-10165–2:logRecord.loggingTime` identifies every instance of the named attributes. By extending the identifiers, the set of identified instances may be restricted. E.g. the name `OSI-10165-2:logRecord.loggingTime!Agent1` identifies all instances of the given type that are administered by the named agent. By adding an instance identifier, it becomes possible to name all instances that belong to an identified set of types and have the same name. E.g. `IAB-MIB2:ifEntry.ifOperStatus!!1` identifies every instance of type "ifOperStatus" named "1".

For most expressions, each evaluation step is carried out with one instance selected from each set identified by a name. In some cases it is desirable that an expression is evaluated for tuples of instances selected from different sets but by the same attribute name, the same agent name or the same instance name. In order to support this, the notation introduced above has to be extended. If attribute identifiers, agent identifiers or instance identifiers are preceded by the word JOIN, they are not taken into account when selecting types or instances. Instead, when they occur in different names present in the same expression, they are interpreted as join variables.

The selection concept

The names introduced in the previous section can render expressions difficult to read, especially, if multiple names occur. This is one justification for the introduction of variables. The domain of a variable is a set of instances. Ist name is an arbitrary identifier. To associate domains with variables, the following constructs are used:

<varDef> ::= [ONE OF] <varScopeDef> (, <varScopeDef>)* [WHERE <logExpr>]

<varScopeDef> ::= <varIdent> IS <setSpec> ((OR | AND) <setSpec>)*

<setSpec> ::= <name> | (SUBORDINATE TO | SUPERIOR TO) (THIS | <name>)

A variable scope definition serves the purpose of assigning a set of instances to a variable. The set of instances is the union (keyword OR) or the intersection (keyword AND) of sets specified at the right side of the keyword IS. A single set may be specified through a name as defined in the previous section. Optionally it is possible to specify as set members, the instances which are contained in at least one of the named instances (keywords SUBORDINATE TO) or the instances containing at least one of the named instances (keywords SUPERIOR TO), respectively. Together with these options the keyword THIS may be used instead of a name. This refers to all instances that are contained in the instance to be mapped or all instances that contain the instance to be mapped respectively. The latter options are needed when defining mappings between attributes within a model.

A variable definition may, besides variable scope definitions, contain a WHERE-clause. The logical expression following the keyword is evaluated once for every variable assignment possible with the domains associated to variables by variable scope definition. Each variable assignment, for which the logical expression can be evaluated to TRUE is retained. The other variable assignments are rejected. In this way, if only one variable scope definition is present in a variable definition, the corresponding variable is associated with the set of all instances for which the expression yields TRUE.

The selection process thus results in a set of variable assignments that may be used when evaluating expressions. If the optional keywords ONE OF are present, one assignment will be selected randomly. The others are not considered for evaluation.

Numerical Expressions

Numerical expressions serve the purpose of calculating attribute values from arbitrary management information. In order to reference instances of management information, variables are used, though direct references to names are allowed, also. To bind variables, the following notation is used:

<numExpr> WITH <varAssignment> (, <varAssignment>)*

To refer to historical information, the keyword BEFORE, followed by a relative time offset may be used. E.g. `ifInOctets!Agent1 BEFORE 2 MINUTES` refers to the value of Attribute "ifInOctets" administered by the agent called "Agent1" which was measured 2 minutes before evaluating the expression.

Simple numerical expressions may include standard arithmetic operators and functions yielding a numeric result. In addition GDMO-M offers two kinds of aggregate expressions. One kind allows numerical aggregation:

numAggrExpr ::= (MAX | MIN | AVG | VAR | SUM | PROD) : <numExpr>

It performs the given aggregation function upon the results of calculating the given numerical expression for every allowed variable assignment. To count the number of instances present in a set represented by a variable or to count the number of assignments allowed for a list of variables, the following statement may be used:

CountExpr ::= COUNT <varIdent> (, <varIdent>)*

The result of evaluating a numerical expression in GDMO-M is the set of values resulting from evaluating the numerical expression once for each allowed variable assignment. Thus, for assigning a value to a non-set-valued attribute, it must be taken care that variables present in the expression for computing this value are only assigned once.

Logical Expressions

In GDMO-M, simple logical expressions yield truth values either through comparison of numerical expressions or through verifying set membership. They may be concatenated through disjunction or conjunction, respectively. Variables are bound using the WITH statement introduced in the context of numerical expressions. The use of quantifiers is possible. The corresponding syntax is:

aggrLogExpr ::= (EXISTS | FORALL) <varAssignment> (, <varAssignment>)* : <logExpr>

A special feature of GDMO-M are timed logical expressions. Each logical expression may be associated with a time variable and a time interval. After evaluating the expression, the set of time intervals for which the expression could be evaluated to TRUE is assigned to the time variable. The time intervals assigned to a time variable are disjunct, i.e. if two time intervals would overlap, they are joined together. By default, evaluation is carried out taking into account every historical version of referenced management information that is available. The result of the overall evaluation is TRUE, if the expression could be evaluated to TRUE at least for one historical version of referenced management information. Else the overall result is FALSE. The syntax of a timed logical expression is:

timedLogExpr ::= <logExpr> ’[’ <timeVarId> [<timeInterval>] ’]’

timeVarId ::= ’t’ <capitalLetter>

The purpose of time variables is to allow the evaluation of relationships between time intervals during which logical statements hold. GDMO-M logical expressions allow to verify, which time relationships hold between the time intervals assigned to time variables. Thus it is possible to verify time relationships between the time intervals during which logical statements are true. E.g. this allows to analyse whether an overload in one switching system occurred during an overload in an other switch or after an overload in the latter system was gone.

Naturally there must exist special operators to express relationships between time intervals. Allen introduced a suitable set of operators with his time logic [Allen 84]. These operators may be used in logical expressions of GDMO-M. The allowed time relationship operators are:

timeRelOp ::=
BEFORE | AFTER | EQUAL | MEETS | OVERLAPS | DURING | STARTS | ENDS.

Since time variables hold sets of time intervals, it has to be specified to which of those intervals a time relationship operator should be applied when evaluating an expression. Selection is specified by using the selection operators ANY, LAST or ALL respectively. The latter operator specifies, that the expression should be evaluated once for every time interval contained in a time variable. In this case, evaluation of the expression yields TRUE if the specified time relationship holds for all time intervals contained in the time variable. The LAST operator is used to express that the expression should only be evaluated for the interval, which has the latest ending point. Since one time variable may not contain overlapping intervals, this selection is unique. Finally, the ANY operator specifies, that evaluation yields TRUE, if the time variable contains at least one time interval for which the expression can be identified as TRUE. The syntax of a logical expression conatining time relationship operators is as follows:

timeRelExpr ::= <timeIntervalSelection> <timeRelOp> <timeIntervalSelection>

timeIntervalSelection ::= (ANY | LAST | ALL) <timeVarId>

The time interval, that may be associated with an expression in addition to a time variable, constrains the period of time for which evaluation of the logical expression should be carried out. Time intervals are specified through their starting point in time and their end point, respectively. Time points are specified in a relative way. An example specification of a time interval is:

SINCE 5 HOURS UNTIL 3 MINUTES

This refers to the time interval which began five hours before evaluating the interval specification and ended three minutes before evaluation. If no starting point in time is given, the interval is considered to begin at the time, when the first values of referenced information have been retrieved. If no end point is specified, evaluation time of the timed logical expression is taken as end point.

5.3 Examples of using GDMO-M

In this section, we will give some examples of mapping rules, illustrating the most important concepts and the usage of GDMO-M. For space reasons we omit the definitions of types used in models as well as the definitions of models for which mapping definitions are given. In all examples but the first, we do not give entire mapping definitions, but concentrate on showing single mapping rules.

At first we show a mapping rule defining the mapping between an attribute with a numerical domain, that may be present in objects of an abstract model and attributes as presented by a management system implementing the IAB-management architecture and offering information as defined in the RMON-MIB [RFC 1757].

```
ethernetMapping MAPPING DEFINITION
MAPS FROM exampleModule:abstractEthernet
MAPS TO IAB-RMON:etherStatsEntry, IAB-RMON:historyControlEntry
CHARACTERIZED BY ethernetMappingPackage MAPPING PACKAGE
        MAPS ATTRIBUTES OF ethernetPackage
        MAPPING RULES
        exampleRule1 MAPPING RULE FOR ATTRIBUTE utilization
        VALUE IS
                (x.etherStatsPkts * 16 + x.etherStatsOctets * 0.8) / y.historyControlInterval
```

```
        WITH x IS IAB-RMON:etherStatsEntry, y IS IAB-RMON:historyControlEntry
        WHERE x.etherStatsIndex = y.historyControlIndex
```

The rule contained in the mapping definition describes how to map between the attribute utilization and some attributes of RMON-MIB. Recall, that assignment of information elements which are taken into account when evaluating the expression is done at object assignment time, only.

The following example of a mapping rule shows the ability of GDMO-M to define mappings in a way that they take into account multiple formats of management information. The given rule may be used in environments where underlying information is provided either by an OSI oriented management system or by an IAB-oriented system. It might as well be used in a heterogeneous environment presenting management information in different formats.

```
exampleRule2 MAPPING RULE FOR ATTRIBUTE compoundSystemState
  VALUE IS
     all-up      IF x = UP WITH x IS IAB-MIB2:ifEntry.ifOperStatus
                 AND y.systemState = running WITH y IS exampleOSIModule:workstationClass
     not-all-up  ELSE
```

The rule shows how to define a mapping for attributes having a qualitative domain.

What follows, is an example of how to use timed relationships in mapping rules:

```
exampleRule3 MAPPING RULE FOR ATTRIBUTE pcsToCheck
  VALUE IS x.name
  WITH x IS exampleGDMOModule:pc, y IS exampleGDMOModule:bridge
  WHERE y.packetRate > 10000 [tX SINCE 24 HOURS]
  AND y.name IN x.connectedTo
  AND (x.noOfUsers = 0 AND x.packetRate > 1000) [tY SINCE 24 HOURS]
  AND tY OVERLAPS tX
```

The rule defines the value of attribute pcsToCheck as the instances of managed object class pc satisfying the following conditions. They must be connected to a bridge, whose packet rate exceeded the value 10000 during at least one time interval in the last twenty-four hours. Furthermore the pc’s packet rate must have exceeded the value 1000 without a user connected during a time period located in the last 24 hours before rule evaluation, too. Last but not least, there must have been be a time interval during which both statements held. Thus, the presented rule assigns to the attribute the names of those pc’s who showed one kind of abnormal behavior during bridge overload.

The last example we present, shows how to use GDMO-M to define model-internal mappings.

```
exampleRule4 MAPPING RULE FOR ATTRIBUTE subnetLoadState
VALUE IS
     noOverload IF      (COUNT x.loadState WITH x IS exampleGDMOModule:system
                                          AND x IS SUBORDINATE TO THIS) = 0
```

```
localOverload IF        (COUNT x
                                WITH x IS exampleGDMOModule:system
                                AND SUBORDINATE TO THIS
                                WHERE x.loadState = overload)
                        / (COUNT x
                                WITH x IS exampleGDMOModule:system
                                AND SUBORDINATE TO THIS
                                WHERE x.loadState = overload) < 0.1
importantOverload IF    (COUNT x    WITH x IS exampleGDMOModule:system
                                    AND x IS SUBORDINATE TO THIS
                                    WHERE x.loadState = overload)
                        / (COUNT x  WITH x IS exampleGDMOModule:system
                                    AND x IS SUBORDINATE TO THIS
                                    WHERE x.loadState = overload) >= 0.1
```

By using the keywords SUBORDINATE TO THIS, a variables scope may be limited to the objects contained in the instance whose attributes are currently mapped.

Discussion

The language presented in the previous sections meets all of the requirements presented in section 5.1. Thus, it is suited for defining mappings between the types used for defining abstract models and the types of underlying management information. Furthermore it may be used to define mappings of types within abstract models.

Because of its underlying concepts, GDMO-M is not limited to defining mappings between underlying management information and managed object classes defined in GDMO. Object classes defined in any other language may be mapped as well.

While automation of mappings from underlying management information to abstract models may be easily realised through a compiler, the inverse mapping is more difficult. If an attribute's mapping is defined by a numeric expression referencing multiple information sources, run-time support is needed in order to coordinate changes to underlying management information.

6 Relation to other work

Some work has been done for defining models that are more abstract than those offered by today's management systems. [Neumair 93] contains definitions of generic models suited to performance management. [Gutschmidt 95] presents abstract models suited for representing different kinds of servers. However, these models are designed for representing single open systems. Thus they cannot represent managed systems containing more than one active network component (open system).

In [Neumair 93] an approach to mapping abstract models to underlying management information is presented. This approach is suited to the models introduced in that work. Only information from one underlying managed object class or structured type may be mapped to an attribute belonging to a class used for defining an abstract model. Thus, it is not possible to use the mapping for mapping information of different sources to one attribute. E.g. it is not possible to define the mapping between an attribute representing a network characteristic and information concerning different components contained in the network.

The language for defining type mappings has not been specified exactly. The examples given in [Neumair 93], show however, that only simple arithmetic expressions or case-statements may be used to define the mapping.

7 References

[Allen 84] J.F. Allen, Towards a general Theory of Action and Time, Artificial Intelligence, Vol. 23, p. 123-154, 1984

[DOMAINS 92] Esprit Project No. 5165, Distributed Open Management Architecture in Networked Systems: DOMAINS Management Language, Document No. D2e V1.0, 1992

[Gutschmidt 95] M. Gutschmidt, Ein Objektmodell für ein integriertes Management von Systemdiensten mit Client/Server-Struktur, Thesis in german language, Herbert Utz Verlag Wissenschaft, Munich, Germany, 1995, ISBN 3-931327-84-1

[IS 9595] International Standard ISO/IEC 9595. Information technology – Open Systems Interconnection – Common management information service definition, Second edition, 1991

[IS 10040] International Standard ISO/IEC 10040. Information technology – Open Systems Interconnection – Systems Management Overview, First Edition, 1992

[IS 10165-2] International Standard ISO/IEC 10165-2. Information Technology – Open Systems Interconnection – Structure of Management Information: Definition of Management Information, First edition, 1992

[IS 10165-4] International Standard ISO/IEC 10165-4. Information Technology – Open Systems Interconnection – Structure of Management Information – Part 4: Guidelines for the definition of managed objects, First Edition, 1992

[Neumair 93] B. Neumair, Objektorientierte Modellierung von Kommunikationsressourcen für ein integriertes Performance Management, Thesis in german language, Technical University of Munich, Germany, 1993

[RFC 1155] Internet Activities Board, Network Working Group, Request for Comments: 1155, Structure and Identification of Management Information for TCP/IP-based Internets, Authors: M. Rose, K. McCloghrie, 1990

[RFC 1158] Internet Activities Board, Network Working Group, Request for Comments: 1158, Management Information Base for Network Management of TCP/IP-based Internets: MIB II, M. Rose (Ed.)

[RFC 1442] Internet Activities Board, Network Working Group, Request for Comments: 1442, Structure of Management Information for Version 2 of the Simple

Network Management Protocol (SNMPv2), Authors: J. Case, K. McCloghrie, M. Rose

[RFC 1757] Internet Activities Board, Network Working Group, Request for Comments: 1757, Remote Network Monitoring Management Information Base, Author: S. Waldbusser, 1995