Proceedings of the *4th Workshop of the OpenView University Association: OVUA'97*,
Madrid, Spain, April 1997

Systems Management with Distributed Objects: Porting SNMP Agents to a CORBA Environment

MNM TEAM

Munich Network Management Team

Alexander Keller
Faculty of Computer Science, Munich University of Technology
Oettingenstr. 67, 80538 Munich, Germany
E-Mail: keller@informatik.uni-muenchen.de

Abstract

The increasing acceptance of CORBA for implementing distributed applications yields the possibility of using this distributed object-oriented communication infrastructure for the management of these applications and the underlying systems as well. It is therefore important to supply CORBA-compliant agents which enable the management of these systems in an efficient manner. Currently, this is not the case: while no such CORBA-compliant management agents are available on the market, one can easily find a multitude of SNMP agents meeting these demands. Thus, establishing a methodology for obtaining the required CORBA-compliant management agents from already existing modular SNMP agents is a necessary task. This paper describes a methodology for migrating already existing modular SNMP-agent code into the CORBA environment and describes its application to a concrete example, namely an agent for managing UNIX workstations. This agent has been developed by the MNM Team and is extensively used. Furthermore, it is sketched out how the transformation overhead can be minimized by the use of standardized translation algorithms and state-of-the-art software development tools.

Keywords: CORBA, Distributed Objects, Distributed Systems Management, Middleware

1 Introduction and Motivation

Apart from the well-known OSI/TMN and Internet management architectures, a third alternative is gaining increasing attention for integrated management: the Common Object Request Broker Architecture which has been standardized by the Object Management Group (OMG). Initially developed for distributed object-oriented programming, the advantages of using CORBA in the domain of network and systems management are more and more recognized.

One reason for this is the fact that today the development of management systems is perceived as a special case of developing large-scale information technology applications. The successful application of new principles from the field of software engineering like object-oriented techniques for the analysis, design and implementation of complex software systems leads to the development of modular, well-structured management applications built from "off-the-shelf" software components. This yields demand for techniques that protect developers, at least to a certain degree, from the heterogeneity of the underlying operating systems. This is particularly important for the development of powerful management software. Another issue is the fact that management architectures like Internet and OSI/TMN require, apart from usual software development knowledge, additional skills in the domain of management information specification. This leads to the situation that the developers (and even their companies) of a distributed application and the corresponding management modules are hardly ever identical. A consequence is that the full range of application-specific APIs useful for management are rarely exploited in depth. The requirement of additional knowledge is due to the fact that traditional management architectures introduce additional complexity by specifying the management interfaces of a managed resource in a special notation (ASN.1 or GDMO) and use dedicated protocols (SNMP or CMIP) for the communication between managing and managed systems. Even worse, these management architectures are totally incompatible; the development of mechanisms for bridging the gaps between them is still a topic of ongoing research. The handling of this heterogeneity introduced by management is often difficult and the requirements of providers of large-scale communication infrastructures are often not met in available products.

The *Common Object Request Broker Architecture (CORBA)* (see e.g. [1], [12]) standardized by the *Object Management Group (OMG)* follows another approach: Initially specified for distributed object-oriented programming, CORBA can be used not only for implementing distributed applications but also for their management. Not only the notation for the definition of application objects and their associated management objects remains the same (namely OMG IDL[1]) but also the access to operational and management data is handled through the same communication mechanism, the ORB. Thus, management becomes an integral part of distributed applications. As a consequence, the development of management software benefits from the broad spectrum of software development tools available on the market because operational and management functionality are modeled and implemented in the same way. Furthermore, studies comparing the information models of the OSI/TMN and Internet management architectures with the OMG object model (e.g. [11]) revealed the suitability of CORBA for the management of end systems and distributed applications.

Despite these advantages, native CORBA-based management agents are still hard to find on the market while SNMP agents are in widespread use. In order to obtain CORBA-compliant management agents from existing SNMP implementations without the need of rewriting the already existing agent code, it is necessary to provide a smooth transition path. This paper will present a novel approach to this problem: it allows the migration of modular SNMP agent implementations into a CORBA environment. Such a CORBA-compliant management agent consists of multiple distributed cooperative managed objects interacting with the managing system via an ORB. The transformation process will be described and demonstrated on a concrete example, namely an agent for the management of UNIX workstations which has been developed by our research group. Furthermore, it is described how the amount of work for the transformation process can be limited through the use of standardized translation algorithms and development tools available on the market.

The structure of the paper is as follows: Section 2 describes the properties of the SNMP agent for UNIX-workstations which serves as a starting point for the transformation. The JIDM algorithm (which has recently been standardized by the Open Group and the NM Forum) for the translation of Internet MIBs into OMG IDL object descriptions is also treated because our approach relies on it. The translation of the SNMP MIB into CORBA object descriptions yields, after some manual treatment, an object model which obviously needs to be enhanced. The reasons and the tools which have been used for the re-engineering of the agent will be described in section 3. During this phase, it is particularly important to have the opportunity of using CASE-tools supporting the cyclic process of analysis and design; it is then also possible to introduce new requirements into the model of the agent system. After this optimized object model has been established, it is necessary to transform it into CORBA-compliant object interface descriptions; they form the base of the implementation which will be sketched out in section 4. Section 5 concludes the paper and gives an overview of further steps.

Our novel transformation approach has proven its usefulness although the boundaries between the different tools are far from seamless at their current stage. Currently, the approach is being applied to other problem domains such as the management of distributed applications (TP-monitors) and network resources like ATM-switches.

[1]The *Interface Definition Language (IDL)* is the standardized language for the specification of CORBA objects

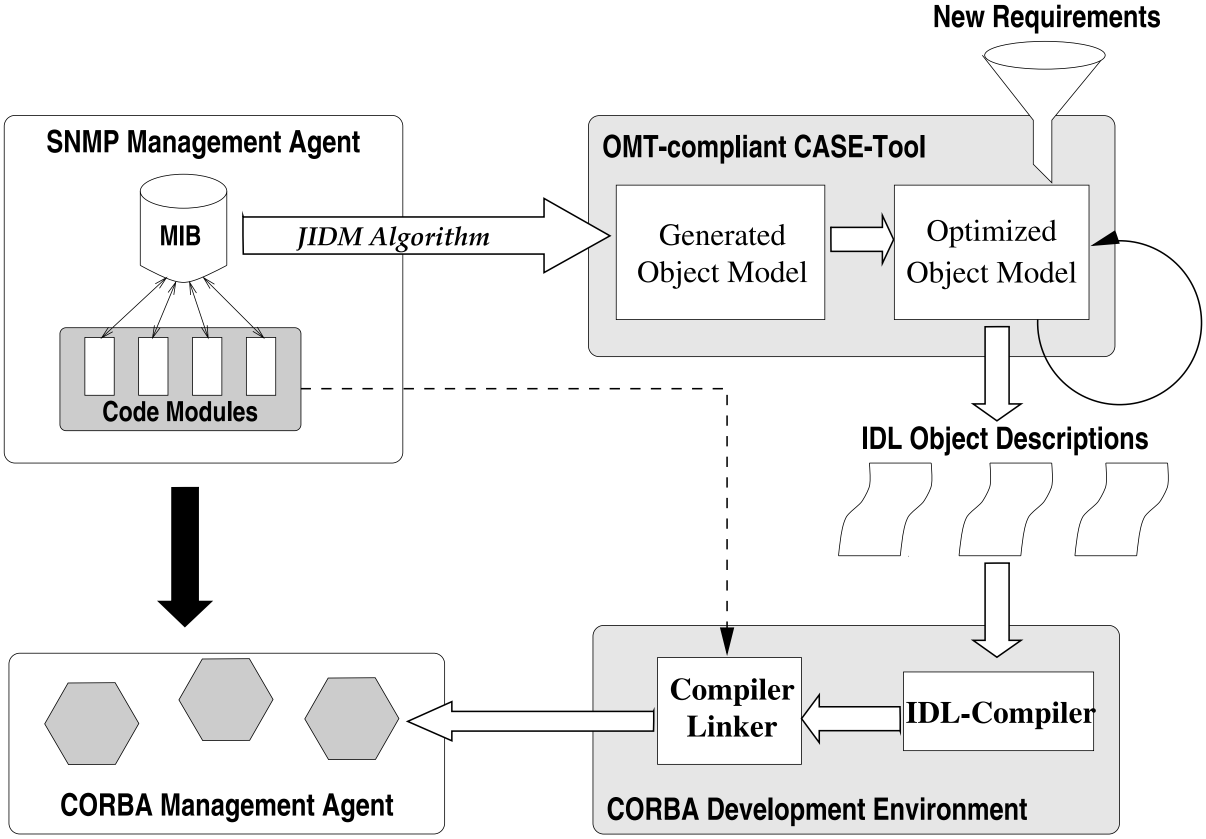

Figure 1: Overview of the transformation methodology

2 Starting point

2.1 An SNMP agent for managing UNIX workstations

In contrast to existing commercial solutions ([4], [8]) generally based on a bottom-up approach, we focused on covering typical tasks of Unix system administrators. This top-down analysis [2] revealed the need for supporting multiple issues: The creation and deletion of user accounts and groups, the management of user quotas for system resources like storage or printer usage, the mount/unmount of filesystems and functions for starting and stopping processes. It is easy to see that the capabilities of the agent are beyond the usual monitoring tasks by permitting the execution of actions on behalf of the systems. The transfer of management information is done through the SNMPv2 management protocol.

The analysis led to the development of a UNIX workstation MIB and to the implementation of a systems management agent running on different platforms (HP-UX, IBM AIX, Sun Solaris, SunOS). The structure of this MIB is depicted in figure 2; the MIB consists of 195 variables and 15 tables and represents, among others, the following components:

- Memory (main memory, swap-space)
- Devices (CPUs, printers, storage disks and filesystems etc.)
- Processes (user processes, kernel processes)
- Users (passwords, groups, quota etc.)

In order to cope with the large heterogeneity of the four supported operating systems and to enable the adaption of the agent to new provider requirements, the agent has been implemented in a modular way:

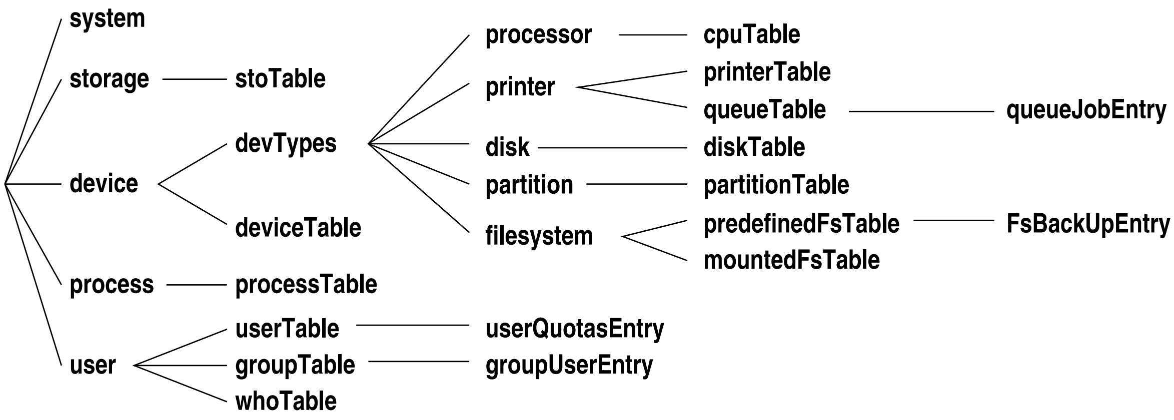

Figure 2: Structure of the UNIX workstation MIB (taken from [2])

Every way of accessing a MIB variable is represented by a different procedure. This means that every MIB variable has been implemented in a separate module and has either one or two interfaces, depending on whether the variable is readonly or read- and writeable. The value of a readonly MIB variable like `sysName` is obtained through the get_sysName procedure; variables which can be read and written are accessed through their appropriate get and set procedures.

2.2 Algorithm for the transformation of SNMP MIBs into the CORBA object model

The mapping of existing management information bases to the CORBA object model implies the development of algorithms for translating the MIB specification languages. Our scenario requires an algorithm for mapping the ASN.1 template language used for the Internet MIB specifications into the OMG *Interface Definition Language (IDL)*. Such an algorithm ([13]) has been developed by the *Joint X/Open NM-Forum Inter-Domain Management Task Force (JIDM)* and was recently adopted by the NM-Forum. The complexity of bridging the different information models may be illustrated as follows: Internet MIBs define the properties of an agent in terms of scalar variables and tables and have no notion of object-oriented concepts like inheritance or polymorphism. In contrast, CORBA agents are defined based on object classes and their associated attributes, methods and relationships. The transformation of SNMPv2 data types, macros and traps into the CORBA mechanisms is described below:

- Every SNMP group becomes an object class; the scalar data types contained therein are transformed into attributes of the object class. The SNMPv2 data types are mapped to their IDL counterparts, e.g. `Integer32` becomes `long`, `DisplayString` and `IpAddress` are mapped to a sequence of octets.
- SNMP tables become object classes, too. Each table entry represents in the OMG model an instance of an object class which is described by an IDL interface. An example shows the application of this rule: If three hard disks are contained in a system, this is represented in the MIB through the existence of a table `storageTable` with three rows. A CORBA-compliant system would instead create three instances of an object class `StorageDevice`.
- Variables specifying table columns become attributes of the corresponding object class.
- SNMP traps are transformed into CORBA events which are implemented through the *Object Event Service* ([7]).

Although the JIDM algorithm is a powerful tool, it is necessary in our case to perform some adjustments because management semantics are often defined in the Internet information model in an implicit way: In

the Internet management architecture, actions on managed objects are performed by assigning a certain value to so-called "pushbutton" variables because the SNMPv2 management protocol has no *action* protocol data unit. The CORBA analogy consists in calling a method of the managed object.
The translation algorithm is unable to perform this mapping because "pushbutton" variables cannot be distinguished from regular variables on a syntactic level.

2.3 Result of the algorithmic transformation: A first object model

The JIDM algorithm described in the previous section serves as a basis for obtaining the first version of a Unix workstation object model. However, two remarks must be made:

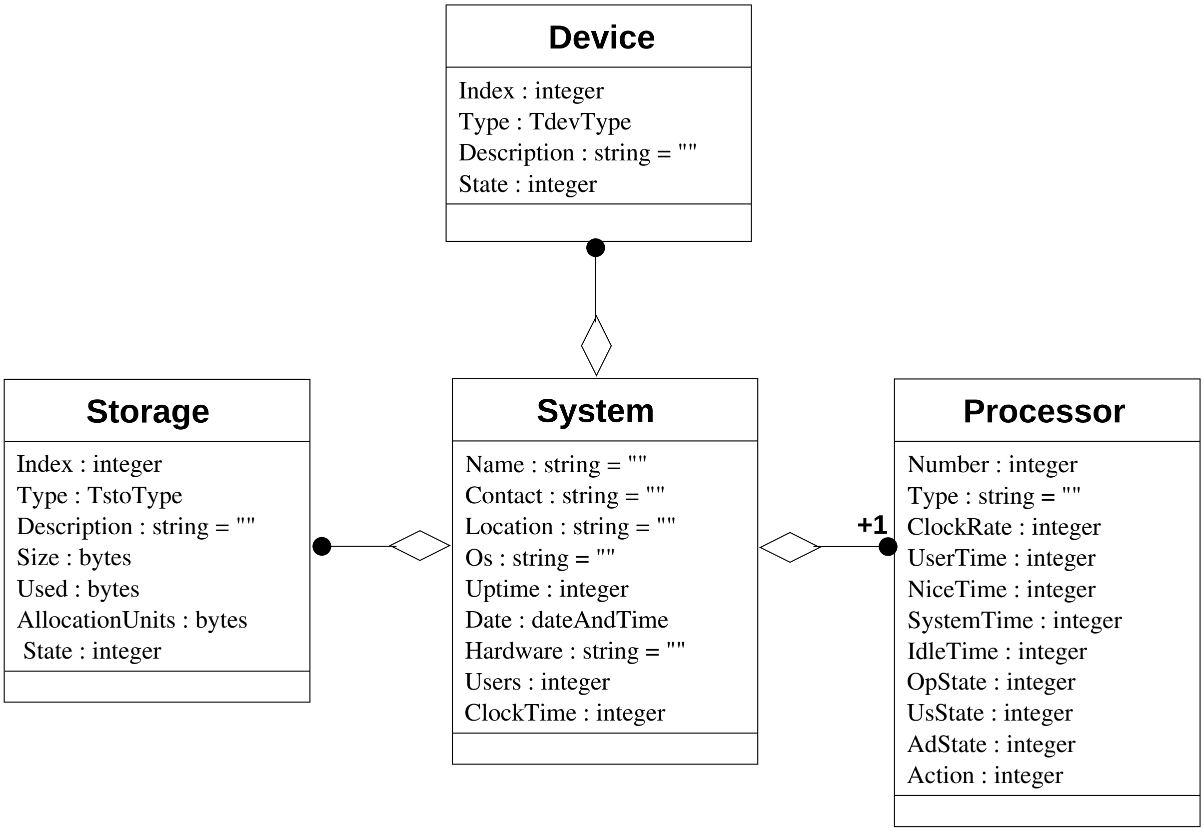

Figure 3: Algorithmic generated object model in OMT notation (partial view)

1. The JIDM algorithm has been developed for implementing management gateways. This implies that a gateway needs to store any information related to SNMP variables (table indices, object identifiers etc.) in object attributes. While this is certainly necessary for management gateways, such an approach is not needed for the redesign of our agent.

2. The target language of the JIDM algorithm is, as already mentioned in section 2.2, OMG IDL. It is absolutely crucial for the further design of the management agent to transform the MIB into a notation which permits further refinements with commercially available CASE-tools. The tool we used (described in section 3.1) uses the *Object Modeling Technique (OMT)* [10]. Thus, the transformation of IDL into OMT had to be done manually but did not take long because the structure and the data types of the object classes and their attributes were already given by the JIDM algorithm.

Figure 3 depicts the first version of the obtained object model.

3 Building an adequate object model

In addition to the transformations cited in the previous section, there are obviously some open issues for the design of the CORBA agent. Remember that the goal of our work is an agent which fulfills *as much as possible* the criteria of an object-oriented system. The critical items to be addressed can be described as follows:

1. **The first object model lacks almost completely (apart from one exception) a hierarchy between the several object classes.**
   The containment relationship between different classes applies only to the `System` class; the model doesn't have inheritance. As a consequence, polymorphism is not supported, too. Thus, three out of the four main principles of the object-oriented paradigm are not fulfilled. This is a consequence of the Internet information model which has no notion of containment and inheritance relationships between object classes.

2. **The "type" variables are not in accordance with the principles of object-oriented design.**
   Examples of such "type" variables in the above described MIB extract are `Type` in the `Storage` class and `Type` in the `Device` class. Although it is possible to distinguish between different kinds of the `Storage` object class (like RAM, hard disks, tapes or diskettes) by setting the `Type` variable appropriately, the specifics of each kind of media are not reflected correctly. In object-oriented modeling, this can easily be expressed through the concept of inheritance. The fact that the Internet information model lacks inheritance is the cause for this anomaly.

3. **The problem of ”pushbutton” variables is still there.**
   The setting of a variable with a specific value results in the execution of an operation; it is therefore difficult to understand the meaning of some attributes. This stems from the fact that SNMP has no *action* protocol data unit.

4. **The data types of the attributes are restricted to the base ASN.1 types.**
   Even attributes with a restricted range of potential values like e.g., `OpState`[2] or `AdState`[3] of the `Processor` object class are bound in the first version of the object model to an integer data type. While this may sound like a ”cosmetic” problem, it is important to restrict the value ranges of attributes with respect to conformance testing.

The only principle of object-oriented design which is met in the first version of our object model is encapsulation. If this object model had been the input to the IDL-compiler, implementation skeletons would have been generated where every attribute is flagged as private. These would have been accessible only via get- or set-operations i.e. the attributes are not accessible by other means than the ones provided by the implementor.

3.1 Tool instrumentation

The deficiencies of the first version of our object model imply massive modifications of its structure. It is also necessary to incorporate new provider requirements not previously foreseen into the process of building an optimized object model. On the other hand, the effort on the side of the developer for implementing the necessary changes should be kept as small as possible.
Thus, we decided to base the re-engineering of our management agent on a commercially available CASE-tool compliant to the OMT design method. Another requirement was the ability of the CASE-tool to generate automatically OMG IDL object descriptions from the object model given in OMT notation. This facilitates the integration with the CORBA development environment.

[2]`OpState` (Operational State) may only take the values 1 (enabled) or 2 (disabled); an enumeration data type is sufficient for this.

[3]`AdState` (Administrative State) has the values 1 (unlocked), 2 (locked) or 3 (shutting down); as in the previous example, an enumeration reflects better the semantics of this attribute.

These requirements were fulfilled by the *Software through Pictures (StP)* CASE-tool [6], which contains several powerful editors for each phase of the software development process. Their navigation features enable rapid prototyping by supporting cyclic analysis, design and implementation steps [9]. The preparation of graphic designs and the application of changes to them in a tabular representation was easily feasible; this is also true for the documentation of the project. Other helpful features were the ability of defining default values for attributes or properties like ”read-only” already in the modeling phase; the fact that the graphical representation of the object model was always clearly arranged made the design easier. The equivalent SNMP MIB covered about 40 pages.

3.2 Steps for optimizing the object model

The observations made at the beginning of this section imply the following rules for optimizing the object model:

- Identical attributes and operations appearing in several different object classes are gathered in a (eventually new) superclass (see 3.2.1).
- Attributes defining the type of different instances of an object class are removed and replaced by appropriate subclasses (see 3.2.2).
- ”pushbutton” variables become methods of the corresponding class (see 3.2.3).
- Relationships between object classes should be as accurate as possible; containment and inheritance are introduced into the model (see 3.2.4).
- New data types (e.g. enumerations) are defined for attributes with restricted value ranges (see 3.2.5).

The results of the application of these rules are described in the following subsections.

3.2.1 New superclasses for common attributes and operations

Several object classes of the first object model contain attributes with identical semantics but different names. These attributes have been gathered in a common superclass and removed from the subclasses which have been derived from it. Typical examples of such common attributes are identificators, name declarations and status attributes appearing, among others, in the `Printer`, `Storage` and `Processor` classes. The role of the superclass has been assigned to the class `Device` which has been renamed in `GenericDevice`. It is now the root of the inheritance hierarchy for several system components. This facilitates further extensions of the object model because these new classes can now be derived immediately from `GenericDevice` and inherit the base properties common to all kinds of components.

3.2.2 New subclasses instead of "type" variables

That optimization is the answer to the second criticism which addresses the fact that different kinds of objects can be instantiated from one object class. Unfortunately, this is contradictory to the concept of object-orientation which implies that instances of the same object class *must* have the same properties. In the first model, it was impossible to assign different properties to different types of an object class; each instance of a class would have had the same structure, independent of its type. An example may illustrate this: It was possible to instantiate objects of different kinds from the `Device` class by assigning different values to the `Type` attribute; each of these components would share only three very generic attributes (*Index*, *Description* and *State*). This is another reason for introducing the virtual i.e., not instantiable base class `GenericDevice` as the root of the inheritance hierarchy (see also 3.2.1). If an object of

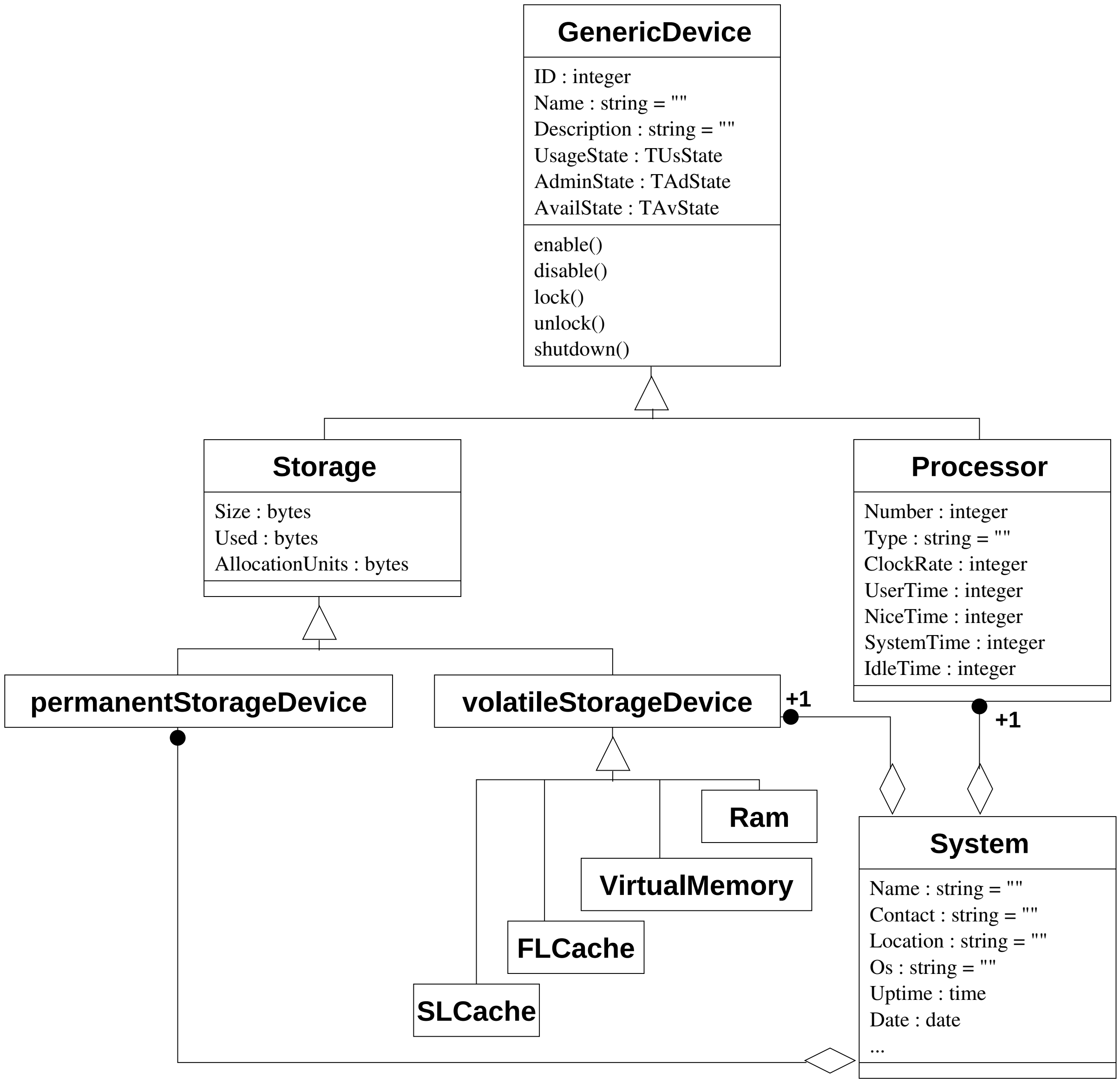

Figure 4: Optimized object model for managing workstations (partial view)

any system component has to be generated, this does not lead to an instance of `GenericDevice` but yields a new instance of the corresponding subclass.

A similar modification affects the `Storage` class: Here, the different kinds of storage (`Ram`, `VirtualMemory`, `FLCache` and `SLCache`[4] could be generated but it was not possible to assign different properties to different kinds of storage devices. Therefore, the new classes `permanentStorageDevice` and `volatileStorageDevice` have been introduced into the model: The former is the superclass for objects like hard disks, magnetic tapes and floppy disks. The latter class encompasses the components which were initially instantiated through the `Storage` class i.e. the new subclasses `Ram`, `VirtualMemory`, `FLCache` and `SLCache`.

[4]**F**irst **L**evel **Cache** and **S**econd **L**evel **Cache** resp.

3.2.3 Operations instead of "pushbutton" variables

The problem of so-called "pushbutton" variables resulting from the Internet information model still exists in the first version of the object model (see figure 3): Operations on a managed object are triggered by assigning a value to the corresponding attribute. This is unacceptable for a model expressed in a powerful notation because this representation permits clearer semantics by having the concept of methods. We therefore introduce a new method for each value that a "pushbutton" variable can take. An example may illustrate this: The five possible values of the attribute `cpuAction` from the class `Processor` lead to the definition of five methods, namely enable(), disable(), lock(), unlock() and shutdown(). The call of the method enable() is now the replacement for the previous value assignment `cpuAction:=1`. As these five operations are needed by many components, they have been added to the base object class `GenericDevice` (see figure 4).

3.2.4 Reality compliant modeling of object relationships

Concerning the relationships between object classes, the first version of the model contains the same deficiencies as the SNMP MIB: There aren't any inheritance and almost no containment relationships. In contrast to this, the goal of object-oriented design is to model the components and their relationships as realistic as possible. The inheritance hierarchy indicates that one component is a refinement of another one, e.g. a storage device is a special kind of device.

The `System` class becomes the root of the containment hierarchy because the end user system is composed of other devices. The definition of aggregation relationships is also straightforward: A `System` contains at least one `Processor` (represented by a 1:n relationship with $n \geq 1$) but any number of `permanentStorageDevices` (in this case, a 1:n relationship with $n \geq 0$). A `Printer`, in contrast, is not part of a `System` but a peripheral device; its connection to the system is expressed by a simple 1:n relationship with $n \geq 0$. Figure 4 gives a overview of the relationships between some object classes.

A special case of relationship can be found between the classes `Filesystem`, `Quota` and `Account`. As the quota is a property of a relationship between a user account and a filesystem, the `Quota` object class becomes an association class of the `Filesystem`—`Quota` relationship (see figure 5).

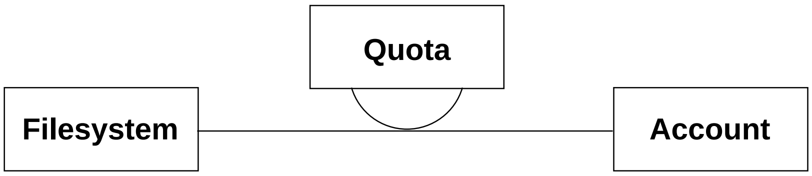

Figure 5: Application of OMT association classes

3.2.5 New types for variables with restricted value ranges

The final optimization step deals with the problem that the ranges of attribute data types resulting from the automatic translation are in general too broad for our purposes. The attribute `OpState` from the `Processor` object class may only take two values in the SNMP MIB: 1 (for enabled) or 2 (for disabled). Due to the lack of more appropriate data types in the Internet management information model, it has been defined as integer. Our object model gives us the possibility of defining an enumeration data type which shows immediately the operational state of a device.

Another example is the attribute `AdminState` (the former `AdState` has been renamed for easier understanding). Here, we defined an enumeration type with four values, namely *unknown*, *unlocked*, *locked* and *shutting down*.

4 Implementing the optimized object model in CORBA

As the CASE-tool provides a mapping from OMT to OMG IDL, the object interface descriptions can be generated automatically. The listing below gives an idea of this mapping by showing the IDL descriptions of the `System` object class.

```
{...}
// stp class definition 108
interface System
{
// stp class members
attribute string Contact;                     // simple read-
attribute date Date;                          // and writable
attribute string Hardware;                    // attributes
attribute string Location;
attribute string Name;
readonly attribute string Os;                 // read-only
readonly attribute time Uptime;               // attributes
readonly attribute long maxProcessNumber;
readonly attribute long maxProcessSize;
attribute sequence<Printer> assnPrinter;      // 1:n association
attribute Process assnProcess;                // simple assoc.
attribute sequence<Processor> aggrProcessor;  // 1:n aggregation
...
};
```

Table 1: IDL interface description of the `System` object class (extract)

One can easily see that attribute properties (like read-only) and their data types defined in the OMT model are transformed into their IDL equivalents. The bottom of the listing contains the definitions of the relationships between different object classes; in this case, information contained in the OMT model is lost by the transformation into OMG IDL.

4.1 Completing the generated IDL interfaces

The loss of information by transforming relationships is a good example for the necessary steps that need to be done manually in order to capture the full semantics of the OMT model in OMG IDL: Although the *Object Relationship Service* ([7]) has been standardized by the OMG, no currently available CORBA implementation contains functionality which helps to enforce the OMT relationship properties during runtime. We therefore had to implement methods by hand to survey the validity of relationships periodically. The method update_processes() checks which processes are currently running on the system.

This example leads us to another issue i.e., the representation of highly dynamic objects like processes: Generally spoken, the CORBA objects representing system resources should be instantiated at the start of the system and deleted when the CORBA runtime environment shuts down. While this is adequate for objects with a low degree of change rates like hard disks and CPUs, this method is unfeasible for administering highly dynamic objects, i.e., objects having a short lifetime like UNIX processes. It is almost impossible to maintain these objects consistent with the real state because that would imply the instantiation of an object at the start of a process and its deletion when the process dies. The solution to this problem consists of doing a snapshot of the current state every time information about processes is required. We achieved this by enforcing the access to `Process` objects through a metaclass `MetaProcess` which called the update_processes method every time when information about processes was required. Any requesting object (usually the monitoring objects from the managing system) gets therefore the correct process state.

Several other modifications were due to the specifics of our development toolkit ([5], [3]) and will be skipped here.

4.2 Reusing the existing agent code

After the completion of the IDL descriptions, the already existing management functionality had to be integrated with the interfaces i.e., the legacy code had to be migrated into the object-oriented environment. The usual technique for achieving this is to encapsulate the legacy code into object classes by means of so-called *wrappers*. It can then be accessed easily through the new capsules by other objects without having the need of modifying the implementation.
Our former SNMP management agent has been implemented in C and has obviously no object-oriented properties. As the OMG has standardized, among others, an IDL–C language mapping, that did not matter. Much more important is the fact that the agent code is well structured and modular. As this was the case for our agent (see 2.1), we did not encounter severe problems while building the new CORBA-compliant management agent.

5 Conclusion and Outlook

Following a practical example, this paper has described the necessary steps for building distributed cooperative CORBA-compliant management objects from an existing SNMP agent implementation. The approach represents a flexible and systematic methodology for the re-engineering of already existing management agents. Three critical factors for a successful migration were identified:

- the modular design of the agent code,
- the availability of standardized mechanisms for transforming the information model from one management architecture into another one, and
- good support by powerful software development tools implementing state-of-the-art OOA/OOD techniques (like OMT).

Several encountered difficulties are due to the fact that currently available CORBA development toolkits lack implementations of already standardized generic services (currently, far more than a dozen CORBAservices have been adopted by the OMG). We believe that a very large part of our manual enhancements may be dropped when CORBA implementations become more mature.
The suitability of CORBA for systems and application management purposes could be demonstrated even if performance and scalability issues make its current use in very large environments prohibitive: This is due to the fact that the interface repositories are implemented as flat files which need to be exported and/or mounted via the *Network File System (NFS)*.
At the current stage of the project, the descriptive power of modern OOA/OOD methodologies has been applied only to static aspects of distributed systems. Further steps consist in analysing and modeling dynamic properties of distributed systems like data and control flows.

Acknowledgements

The author wishes to thank the members of the Munich Network Management (MNM) Team for helpful discussions and valuable comments on previous versions of the paper. The MNM Team directed by Prof. Dr. Heinz-Gerd Hegering is a group of researchers of the University of Munich, the Munich University of Technology, and the Leibniz Supercomputing Center of the Bavarian Academy of Sciences.

References

[1] The Common Object Request Broker: Architecture and Specification. OMG Specification Revision 2.0, Object Management Group, July 1995.

[2] M. Gutschmidt and B. Neumair. Integration von Netz- und Systemmanagement: Ziele und erste Erfahrungen. In *Proceedings der 3. Fachtagung Arbeitsplatzrechensysteme (APS'95), Hannover*, May 1995.

[3] IBM Corporation. *SOMobjects Developer Toolkit Programmer's Guide Volume 2: Object Services*, March 1996. First Edition.

[4] IBM Corporation, International Technical Support Organization, Research Triangle Park, NC 27709-2195. *IBM Systems Monitor: Anatomy of a Smart Agent*, December 1994. Order Number: GG24-4398-00.

[5] IBM Corporation, International Technical Support Organization, Research Triangle Park, NC 27709-2195. *SOMobjects: A Practical Introduction to SOM and DSOM*, July 1994. Order Number: GG24-4357-00.

[6] Interactive Development Environments, Inc. *Software through Pictures/Object Modeling Technique: Getting Started with StP/OMT*, 1996. Release 3.

[7] CORBAservices: Common Object Services Specification, Volume 1. OMG Specification, Object Management Group, March 1996.

[8] HP OpenView Operations Center Concepts Guide. User manual, Hewlett Packard, 1993.

[9] Robert M. Poston. Automated from Object Models. *Communications of the ACM*, September 1994.

[10] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall International, Inc., 1991.

[11] Tom Rutt. Comparison of the OSI Management, OMG and Internet Management Models. A report of the joint x/open-nm forum inter-domain management task force, AT&T Bell Laboratories, March 1994.

[12] Jon Siegel. *CORBA Fundamentals and Programming*. John Wiley & Sons, Inc., 1996.

[13] Inter-Domain Management Specifications: Specification Translation (Final Sanity Check Draft). Preliminary Specification Pxxx, X/Open Ltd., September 1996.