Integration of WBEM-based Management Agents in the OSI Framework

O.Festor, P.Festor,N.Ben Youssef
LORIA - INRIA Lorraine
Technopôle de Nancy-Brabois
Campus scientifique
615, rue de Jardin Botanique - B.P. 101
54600 Villers Lès Nancy Cedex,
France
{festor,benyou}@loria.fr

Laurent Andrey
BULL – DYADE – LORIA
615, rue de Jardin Botanique - B.P. 101
54600 Villers Lès Nancy Cedex,
France
andrey@loria.fr

Abstract

In this paper, we propose a set of mappings and an implementation of an integration agent allowing WBEM-based agents implementing a CIM information model to be managed by OSI-based management platforms and applications. Extending existing integration approaches, this paper provides three original items that are the support of the CIM meta-model in an OSI agent, the mapping of relationships onto GRM specifications as well as a full Java-based implementation of the Q.adapter.

Keywords

CIM, CMIS, Gateway, GDMO, GRM, Java, Management Integration, OSI, TMN, WBEM

1. Introduction

Despite many years of standardization, systems, network, and service management, hereafter called management, still has to focus on integrating heterogeneous management approaches. Should standardization be blamed for this? Certainly not. First, the standardization efforts have contributed to reduce drastically the number of proprietary approaches to management. Second, standardization has enabled the wide acceptance of the main paradigms, which form today's basis of management architecture, i.e. the manager/agent paradigm, the notion of resource independent management information model,

Reasons that motivate the survival of multiple approaches in the management field are manifold. First some approaches are better suited to some management levels than others in terms of complexity, expressive power, available development tools, costs and available information models (e.g. SNMP in the domain of equipment and Internet management, OSI in the domain of telecommunication network management). Second, the large investments, both financial and human, made on all these approaches make them become tomorrow's legacy systems. Third, the rapid growth of distributed object technology opens promising perspectives to the

management community. Especially the combined use of CORBA and Java technologies seems particularly appropriate for the deployment of tomorrow's service management solutions.
Current TNM architecture copes with integration using mediation devices and any kinds of Q.adaptor [1].
One of the approaches that are subject to integration is Web-Based Enterprise Management (WBEM) [2, 3]. Launched in 1996, this approach gains acceptance in the desktop and LAN management community and agents based on this paradigm start to become available on several systems (NT 4.0 SP5, NT5.0)..
In this paper, we present a Q.Adapter (also called integration agent) which allows WBEM-based agents to be managed by OSI-based management systems. A WBEM-based agent is defined here as an agent that implements the Common Information Model (CIM) and which is accessible through the Hyper-Media Management Protocol (HMMP [3]). WBEM-based agent and CIM-agent are used synonymously in the remainder of this paper.
So far, few efforts have been ported on integrating those approaches and most existing proposals are concerned with accessing TMN-based management systems through WBEM managers. This integration way is needed when a virtual private network is managed using WBEM (end stations, management stations) but long distance links management comes as a TMN operator service.
In our approach, we look at the dual problem: accessing WBEM-based agents from TMN-based management. This scenario occurs when a service provider manages parts of a customer network or when a TMN-based Management System of an enterprise has to address some CIM-based systems of its own information system (i.e NT database servers). To this end, we propose mappings for information models, naming issues and services from WBEM to TMN. We also present an implementation of the Q.adapter in a full Java environment using especially a Java CMIS interface as well as information model management tools developed in our group.
The remainder of this paper is organized as follows. Section 2, provides a short summary of the tremendous work already achieved on management integration. Section 3 compares the features of the WBEM and OSI approach that are relevant in our integration proposal. Section 4 details the information model and Management Information Base (MIB) architecture mappings for the Q.adapter. Section 5 presents the CMIS/HMMP service mapping. Section 6 provides a description of the implementation of the Q.adapter that realizes the integration. After a short description of the CMIS Java API developed in our group and the MOF to GDMO translator, it details the architecture of the agent and outlines encountered problems. A conclusion summarizes the paper then sketches some open issues and future work.

2. Related work on management integration

Within the last years, many efforts have been spent on analyzing integration methods and providing various types of management gateways. [4] gives a classification of the various methods and related implementation choices.
Several architectures and algorithms for combining SNMP and OSI-based management have been proposed. In [5], a SNMP/OSI gateway is proposed. In this

approach, a mapping from SNMP Structure of Management Information (SMI) to GDMO is advocated and a global network information model is proposed. The concept of *stateless* and *stateful* agent presented in this approach is reused in our proposal. A generic approach to SNMP and OSI management integration is also presented in [6]. In [7], an integration mechanism is proposed at the agent level allowing an agent to respond to both CMIS and SNMP-based requests. An implementation of an OSI/SNMP Q.Adapter based on the IIMC [8-10] translation algorithms and providing useful information on the agent architecture is detailed in [7].
In the domain of legacy systems integration, [11] addresses issues in managing TL.1 devices through SNMP.
A big work on management integration was achieved by the JIDM [12]. There the focus was placed on mapping both SNMP and OSI-management paradigms (i.e. specifications and services) to CORBA IDL and conversely. In our approach, we have reused some of the principles defined in this proposal, especially some of the naming conventions.
In the domain of WBEM and OSI integration, very few documents have been published so far. The main contribution is Vertel's white paper on WBEM and TMN [13]. In this proposal a two way mapping is envisaged but the emphasis is made on accessing OSI-based agents from WBEM-based managers.
As we will point out in the next sections, many of the concepts developed in the above proposals have been reused in our work and new ones will be outlined.

3. A comparison of OSI and WBEM-based Management

WBEM and OSI management have many common concepts, despite their realizations differ both in terms of denomination and details. Both approaches use a Manager/Agent paradigm. One agent maintains a MIB: a set of Managed Objects (MO), abstraction of the actual resources. A management service and protocol give access to the agent's MOs.
Both approaches use an object-oriented paradigm for information modeling. However the information model approaches differ on several points summed up in Table 1.

Points	**OSI**	**DMTF**
Specification languages	-Guidelines for the Definition of Managed Object (GDMO) [14] (MOC, MO instance) -General Relationship Model (GRM) [15] -Abstract Syntax Notation 1 (ASN.1) [16] (data-type)	Managed Object Format (MOF) [17] (MOC, MO instance, data-type)
Specification unit	Module (with OSI global naming)	Schemas
Inheritance	Multiple	Simple
Attribute scope	Outside MO class	Directly defined in MO

	Grouped in packages optionally added to class	class context
Data- type (attributes, parameter, reply)	All possibilities of ASN1 (list, sequence…)	Limited set of basic data-type (integer, boolean string)
Basic containment relation and naming	Hierarchical Name binding between MO **instances** using one attribute per instance Instance Distinguished Names	Name-space (Hierarchical directory structure) for MOC and MO instances. Many *key* attributes per instance
Relationship between MOs	- pointer attribute - GRM class & instance - management action and/or operation	-special managed objects containing references to other objects (*association* qualifier)
Specifications Repository	Dedicated MIB definition and agent [1]	MOF MOC definition are stored in name space
Management service/protocol	CMIS/CMIP [18] Basic service for **instances**: manipulation, creation, and deletion. Most of the services can apply on set of instances specified by scoping rules and filters.	HMMP Basic service for instances and classes manipulation, creation, deletion. SQL like query language (HMQL)
Event	GDMO specification Management service available (Event Report service)	

Table 1. OSI vs. DMTF approaches comparison.

4. Information model mapping

In order to provide an automated translation from CIM-based specifications expressed in the MOF notation to GDMO/ASN.1 specifications, one must define mapping rules for all components of the CIM model. In this section, we detail those mapping rules. After a description of the chosen mapping philosophy, we focus on the generic GDMO templates necessary for the translation. We then detail the mapping algorithms for managed object class specifications. These algorithms are refined with the consideration of MOF qualifiers and MO instances as well as meta-model constructs found in CIM-specifications. Finally a short example of an automated translation is given.

4.1 Mapping Goals

The goal of the integration agent is to provide an OSI view of CIM-based agents to OSI-based management applications and managers as illustrated in Figure 1.

The integration agent must be able to handle multiple CIM-agents and for each CIM-agent, mappings must be able to allow the visible OSI MIB to offer a hierarchical view of the MO's similar to the hierarchy based on namespaces in the CIM-agents. An illustration for this requirement is given in Figure 1 where we have expanded a CIM-agent (**CIM-Agent 2**) whose MIB contains a three level hierarchy of namespaces, i.e. `root`, `default` and `PC`. In each level, MO classes as well as instances can be found. We have shown the expected associated OSI MIB in the WBEM-OSI integration agent. There, MOs are organized in a similar way using a name-binding containment relationship.

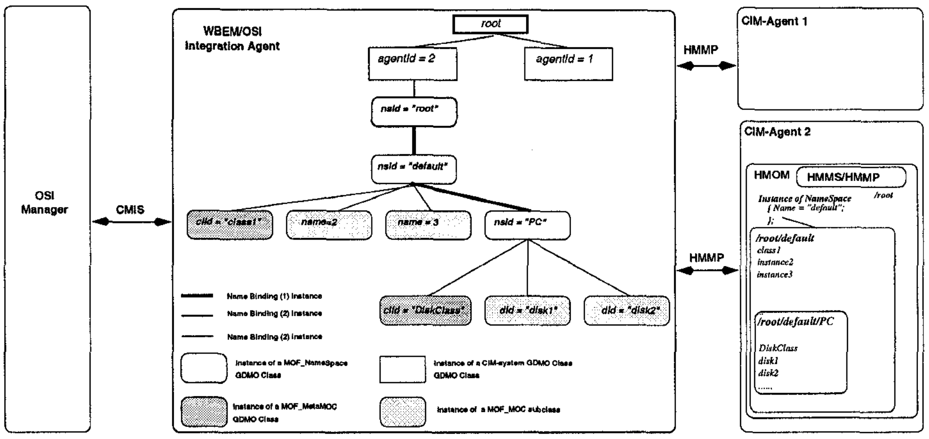
OSI Manager
CMIS
WBEM/OSI Integration Agent
root
agentId = 2
agentId = 1
nsId = "root"
nsId = "default"
clId = "class1"
name=2
name = 3
nsId = "PC"
clId = "DiskClass"
dId = "disk1"
dId = "disk2"
Name Binding (1) Instance
Name Binding (2) Instance
Name Binding (2) Instance
Instance of a MOF_NameSpace GDMO Class
Instance of a CIM-system GDMO Class
Instance of a MOF_MetaMOC GDMO Class
Instance of a MOF_MOC subclass
HMMP
HMMP
CIM-Agent 1
CIM-Agent 2
HMOM
HMMS/HMMP
Instance of NameSpace { Name = "default"; };
/root
/root/default
class1
instance2
instance3
/root/default/PC
DiskClass
disk1
disk2
......

Figure 1. Overall objective of the integration agent

The main objective of our work was to define the algorithms and validate them through an implementation allowing automated generation of the integration agents. Referenced GDMO classes, name-bindings, realization details are presented in the next sections.

4.2 Generic GDMO constructs

To define our mapping we have specified three generic GDMO classes and associated Name-Bindings. These MO classes are illustrated in Figure 2.

The first generic GDMO class used in our mapping is the `CIM_System` class. It derives from the `''ISO 10165-2'':Top` class. This class contains one package which itself contains two attributes. One attribute defines the corresponding CIM-agent name or IP address, the second attribute defines the status of the agent (`up`, `down`). This class is similar to the `snmpSystem` class defined in the IIMC SNMP to OSI translation [8-10]. This class is instantiated every time a CIM-agent is registered within the integration agent.

The second generic class used in our approach is the `MOF_NameSpace` class. This class represents an item of a namespace in the CIM approach, i.e. one basic component of a namespace. This class contains only one attribute, which is the name of the component, e.g. `Root` or `Default` namespaces from the example in Figure 1.

The third generic class is the MOF_MetaMOC class. This class contains attributes that represent a MOF MOC specification, i.e. qualifiers, attributes, methods. This class is used in a similar way as the Management Knowledge Management Function [19] in the OSI framework enabling MOF MOCs to be accessible by management applications. In other words we implement MOF class model using GDMO templates.

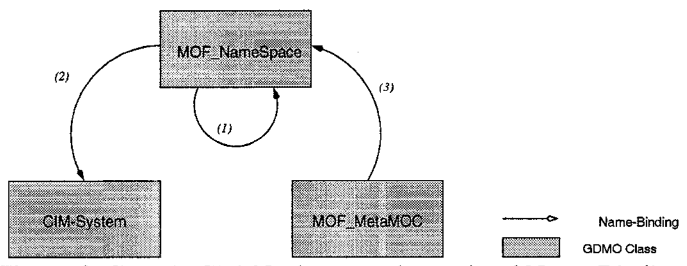

Figure 2. Generic GDMO classes and associated Name-Bindings

Between these three classes, three name-bindings have been defined. Name-binding *(1)* links a superior MOF_NameSpace to a subordinate MOF_NameSpace. This name-binding reproduces in the OSI MIB the containment relation between to intermediary directories of the CIM MIB. Name-binding *(2)* is defined between the MOF_NameSpace class in the subordinate role and the CIM_System class in the superior role. In the OSI integration MIB this name-binding links the representation of a CIM agent to the root of the representation of its name space. The last predefined name-binding *(3)* links one MOF_MetaMOC class in the subordinate role to a MOF_NameSpace class in the superior role, reflecting that MOF specifications in the CIM-agent are logically found within name spaces. In other words this name-binding links CIM leaves (classes, instances) representation to name space representation.

4.3 Implementation of MOF classes using GDMO

Mapping a managed object class defined using the MOF notation into a set of GDMO templates is straightforward. This is due to the fact MOF constructs are simpler than GDMO ones except for the qualifier constructs. Basically, the algorithm works as follows:

- A MOF schema is mapped onto a GDMO module and all definitions found in this schema are labeled by the associated name in GDMO ;
- To each class defined in the MOF framework, a GDMO MOC having the same name and one GDMO package which contains all attributes, actions and notifications defined for this class into the corresponding MOF template are defined ;
- To each MOF class, one associates a name-binding *(3)*, with the corresponding GDMO class as the subordinate object and the MOF_NameSpace class as the superior object ;
- To each attribute in a MOF class, a GDMO attribute template is built and the MOF type is mapped onto a corresponding ASN.1 type. This attribute is then added to the above mentioned class package. Naming of the attribute, i.e. its

label, is the concatenation of the class name and the attribute label, avoiding conflicts between attributes with the same name defined in several MOF classes ;
- Each method is translated into a GDMO action. The method parameters and return type are translated into corresponding fields and reply in the GDMO action information syntax.

Mapping of MOF data types to ASN.1 is also straightforward since MOF data-types are basic compared to the power of ASN.1. For this task we have used the same mappings as those defined in [13]. These mappings have however been extended with the support of constraint qualifiers (e.g. maximum string size) which are mapped to ASN.1'92 constraints.

4.4 Mappings for qualifiers

The CIM Structure of Management Information (SMI) allows the definition of qualifiers, which can be associated to specification components to extend their semantics. These qualifiers have a direct impact on our mapping algorithm. In our approach, we cope with three basic qualifiers: *key*, *read*, *writes*. These apply to attributes.

Key qualifier marks which attributes are used to uniquely identify instances in a CIM name space. For a given MOF MOC if only one attribute is a key then this attribute is used in the `with attribute` clause of the name-binding *(2)* generated for the class. If multiple attributes are key in a MOF class, the mapped GMOD MOC is featured with an extra attribute: concatenation of all mapped key attributes. This new attribute is the one used in the name-binding *(2)*.

Table 2 shows *read*, *write* attribute qualifiers mappings. An absent *read/write* qualifier is assumed to be true.

MOF qualifiers		GDMO attribute properties
read	*write*	
true	true	GET-REPLACE
true	false	REPLACE
false	true	GET
false	false	Warning. Ensure "private" attribute in OSI agent implementation.

Table 2. MOF qualifiers mappings.

We also handle *association* class qualifier. This qualifier allows a MOF class to use reference attributes to implement association between instances. Each time an *association* qualifier is found in a MOF class definition, we create a GRM relationship and the regular `MOF_MetaMOC` GDMO class. Reference attributes (attributes with distinguished name as type) are freely used in GMDO class definition. All other MOF qualifiers are integrated to the quite informal GDMO `behaviour` template.

4.5 Dealing with MOF instances and MOF specifications in the MIB

As already mentioned in section 3, each CIM agent maintains within its management information base both MO instances and MO class definitions, e.g. the MOF specification of the class *DiskClass* is maintained in the *root/Default/PC* namespace of the CIM-Agent 2 in Figure 1. To provide the same feature in the associated Q.Adapter, every time a MOF specification is found in a CIM-agent, an instance of `MOF_MetaMOC` is created in the integration agent. Within this instance, one finds all components of the class specification, i.e. its qualifiers, name, attributes, operations. This approach is similar to the Management Knowledge Management Function [19] in the OSI approach where one can instantiate in the OSI-agent the GDMO specifications of the contained objects and components (GRM specifications, name-bindings...).

Each time an instance of a MOF class specification is found in the CIM-MIB (e.g. *disk1*, *disk2* of class *DiskClass* in Figure 1), a corresponding instance of the associated GDMO class is created. As it will be shown in the next sections, this can be achieved in two ways: generating CMIS Create calls, which will be invoked on the integration agent or by implementing this function directly in the integration agent together with a discovery facility.

4.6 A mapping example

Based on the algorithms presented in the previous sections, we illustrate here the result of an automated mapping for the MOF-based Managed Object class specification given in Example 1. For size reasons, we took a simplified version of the `ManagedSystemElement` (line 3) class from the CIM Core schema as specified in line 2. This class has two attributes, one used for naming (attribute `Name` line 8) and one used to state if the element is installed or not (attribute `Installed` of type `boolean` on line 10).

```
1 [ Description ("The ManagedSystemElement class is the
                 base class ..."),
2 Schema ("Core")]
3 class ManagedSystemElement
4 {
5 [Description ("The Name property defines ..."),
6 Key(true),
7 Write(false)]
8 string Name;
9 [Description ("A boolean ....")]
10 bool Installed;
11};
```

Example 1. A simple MOF class.

Example 2 shows the GDMO mapping of Example 1. There is one description qualifier defined for the class (Example 1, line 1 clause `Description(``...'')`). According to the mapping of qualifiers, the content of this description qualifier, is mapped onto the `behavior` clause of the package associated to the GDMO class (Example 2, lines 26,27).

Since the MOF specification is made in the context of a schema called `Core`, the corresponding, GDMO specifications are grouped in a GDMO module, which has the same name (Example 2, line 1).

The MO attributes defined in the `ManagedSystemElement` class are mapped onto GDMO attributes (Example 2, lines 16 to 24) and those attributes are registered in the GDMO package associated with the MOC (Example 2, lines 12 and 13). To define which operations are allowed on the attributes, a parsing of the attribute's qualifiers is required in the MOF specification.

```
1 MODULE ''Core'';
2
3 ''Core'':MOF_Core_ManagedSystemElementClass MANAGED OBJECT CLASS
4 DERIVED FROM MOF_MOC;
5 CHARACTERIZED BY ''Core'':MOF_Core_ManagedSystemElementPackage;
6 BEHAVIOUR
7 ''Core'':MOF_ManagedSystemElementBehaviourPackage;
8   REGISTERED AS { schemas(2) classes 1 };
9
10 Core:MOF_ManagedSystemElementPackage PACKAGE
11 ATTRIBUTES
12  ''Core'':MOF_NameAttribute GET,
13  ''Core'':MOF_InstalledAttribute GET-REPLACE;
14 REGISTERED AS { Schemas(2) packages 1 };
15
16 ''Core'':MOF_NameAttribute ATTRIBUTE
17 WITH ATTRIBUTE SYNTAX GraphicString;
18 BEHAVIOUR ''Core'':MOF_NameAttributeBehaviourPackage;
19 REGISTERED AS { Core attributes 1 };
20
21 ''Core'':MOF_InstalledAttribute ATTRIBUTE
22 WITH ATTRIBUTE SYNTAX BOOLEAN;
23 BEHAVIOUR ''Core'':MOF_InstalledAttributeBehaviourPackage;
24 REGISTERED AS { Core attributes 4 };
25
26 ''Core'':MOF_ManagedSystemElementBehaviourPackage BEHAVIOUR
27 DEFINED AS "Description: The ManagedSystemElement class ..." ;
28 ...
```

Example 2. GDMO mapping of example 1.

As explained in §4.5 a name-binding (3) must be created to anchor the class specification in the containment tree. Example 3 outlines such a binding.

```
1 ''Core'':nameBinding1 NAME-BINDING
2 SUBORDINATE OBJECT CLASS ''Core'':MOF_Core_ManagedSystemElementClass ;
3 NAMED BY SUPERIOR OBJECT CLASS ''Core'':MOF_NameSpace;
4 WITH ATTRIBUTE ''Core'':MOF_NameAttribute;
6 BEHAVIOUR
7 ''Core'':nameBinding1Behaviour;
8 REGISTERED AS { schemas(2) name-binding 1 };
9 ...
```

Example 3. Generated name binding for example 1.

The class is defined as the subordinate and the superior class in the name-binding is of type `MOF_NameSpace`. The attribute used for naming instances of the subordinate class is the MOF_NameAttribute chosen because the `key` qualifier was associated to the attribute definition in the MOF specification.

5. Service mapping

This section presents the mapping of CMIS operations onto corresponding HMMP requests through the Q.Adapter MIB. Due to the *stateful* nature of the Q.Adapter, **Get** operations are either mapped onto corresponding HMMP **Enumerate**, **Get** or **Query** operations or not propagated to the CIM-Agent at all. This depends on the nature of the MOs selected in the scope of the CMIS **Get** operation. If operations related to the retrieval of MOF classes, then the request is not propagated to the CIM-agent and the response is built directly from the OSI MIB. If the CMIS **Get** operation addresses only MO instances, then depending on the scope and filters, the agent issues a combination of HMMP **EnumerateInstance** and **GetInstance** HMMP operations to the CIM-agent.
All CMIS **Set** operations are mapped to HMMP **Put** operations. For each affected MO in the OSI MIB, if the MO represents a MOF class, then a HMMP **PutClass** is issued, otherwise, a **PutInstance** operation is built and sent to the CIM-agent
Invocation of actions through the CMIS **Action** request, are mapped onto HMMP **ExecuteMethod** service invocation. CMIS **Delete** and **Create** service operations are mapped respectively onto HMMP **Delete** and **Create** operations. The subtype of HMMP operation chosen for the operation **Class** or **Instance** depends on the type of the concerned OSI MO in the MIB. If one wants to create or delete an instance of a `MOF_MetaMOC` class, then an HMMP class operation is built. Otherwise, a **CreateInstance** respectively **DeleteInstance** HMMP request is built. Note that one CMIS **Create** request always results in one HMMP **Create** operation and that one CMIS **Delete** operation may generate several HMMP **Delete** operations depending on the CMIS scope and filters.

6. Mapping the other way around

The main goal of our project was not concerned with accessing OSI-based agents through WBEM managers, which requires the full mapping of GDMO/ASN.1 to MOF. This can be achieved in the same way it is done within the JIDM for the GDMO/ASN.1 to IDL mapping. Considering the GDMO to MOF mapping, translation rules should be similar to the GDMO to IDL ones with a first parsing phase in the GDMO specification whose goal is to resolve multiple inheritance and a second phase of MOF specification generation. There an extensive use of new qualifiers such as `package<label>` who tells to which package an attribute belongs should be made.
At the service level, mapping is also quite easy, Basic HMMP operations have their counterparts in CMIS and more sophisticated operations such as Query can be mapped to basic CMIS services enhanced with scoping and filtering selection facilities.

7. Implementation experience

In order to validate the mappings proposed so far, we have implemented a full Q.Adapter. In this section, the whole software developed for this purpose is presented. Figure 3 shows the overall architecture of the integration agent environment.
As illustrated in the architecture, several tools and APIs are used from which most have been developed in our group for this project. The whole tool-set is available in Java and most of the components are already freely available on the WEB [20].
Building a WBEM/OSI integration agent requires the availability of the MOF specifications defined in the CIM-agent. These specifications are given to the *MODERESJava* toolkit that is responsible for building the associated GDMO, GRM and CMIS scripts according to the mapping principles.

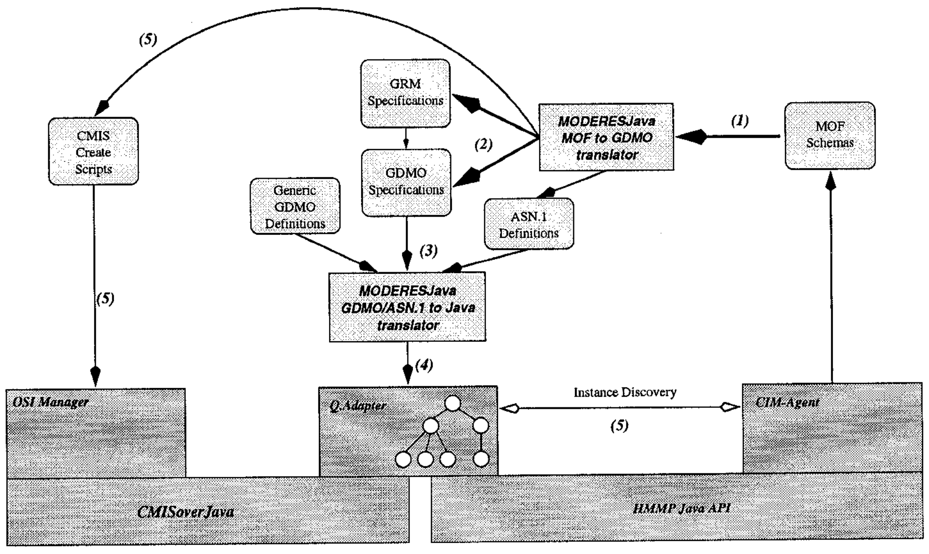

Figure 3. Initialization and operation of the Q.Adapter

The resulting GDMO and GRM specifications are again given to the *MODERESJava* toolkit, which builds associated Java classes. Those classes are bound to the generic Java classes of the Q.Adapter to form the final integration agent. In the next sections, we will focus on three features of this toolkit, namely the Java CMIS communication interface, the *MODERESJava* information model toolkit suite and the integration agent itself.

7.1 The communication interfaces

In order to implement the Q.adapter in Java, we needed two Java-based communication interfaces: one offering an HMMP client (manager) role API and one offering an agent-side CMIS API. The first (HMMP) API was once upon a time

provided within the WBEM SDK distribution and provides a set of packages allowing the manipulation of CIM-based classes and instances as well as HMMP operation invocations.
Concerning the CMIS Java API, none was available when we started the project. We thus decided to build a CMISJava API for this purpose. The result, called *CMISoverJava* is a full object-oriented CMIS API (both manager and agent service interfaces are defined). *CMISoverJava* is close to the TMN/C++ [21] specification except for the asynchronous event model built over the Java AWT event model.
Each CMIS invocation (Request, Indication, Response, Confirm) is defined as a Java class. To each operation invocation, the programmer can associate a listener, which will be informed when a response or a framework error arises. On the manager side, an application which wants to issue CMIS request, first binds to the framework (a partial abstraction of the OSI association), creates request instances, sets the request parameters and issues the request by calling the `doIt()` method (see a Get Request example on the left side of figure 4).

```
public class testGet
{
 public static void main(String[] args)
 {
//defining the request
GetRequest get = new GetRequest;
// defining a listener
MyGetListener myGetListener
    = new MyGetListener();
// Defining ASN.1 values
OctetString str = new OctetString("21.3.2");
RDN = new RelativeDistinguishedName();
 .....
// assigning the request parameters
get.setBaseObjectClass(
                new ObjectClass(oid1));
get.setBaseObject(oi);
get.setScope(
  Scope.BASE_TO_NTH_LEVEL(2));
get.addGetListener(myGetListener);
try{ // sending the request over the stack
 get.doIt();
}
catch (RuntimeException e){
System.out.println(e.getMessage());
System.exit(1);
}}} // testGet
```

```
public class MyGetListener
          implements GetListener
{
 public void complete(GetCompleteEvent pCe)
 {
 System.out.println("End of responses");
 }
 public void reply(GetReplyEvent pGr)
 {
 if (pGr.getReply().getFMKErrors() == null &&
pGr.getReply().getCMISErrors() == null )
 { // printing all attributes received in the
      response
 AttributeList list =
            pGr.getReply().getAttrList();
 for (Enumeration e = list.getElts() ;
      e.hasMoreElements() ;)
 { System.out.println(e.nextElement().toString());
 }}}
} // MyGetListenener
```

Figure 4. A sample CMIS Java code

The right part of Figure 4 contains an example of a listener defined for the Get Request. Each listener contains two methods:

- The `reply` method, which is called each time a response to a previously issued request, is received. The response event contains the received decoded CMIS response ;

- The `complete` method that is called when a CMIS call is complete, i.e. all responses have been received, no confirmation was required or a framework error occurred.

An application or agent developer cans subclass all listeners to define their own treatment of incoming confirmations (manager side) or indications (agent side).

Notifications are handled on the manager side on an application subscription basis. Each application, which wishes to receive notifications, builds a notification listener and registers this listener by the *CMISoverJava* manager handler. On the agent side, event-report issuance is done in the same way as request invocations on the manager side.

ASN.1 types and values are predefined classes in the *CMISoverJava* package. Each Java ASN.1 class inherits from a generic *CMISType* and provides several access methods. In the RMI implementation, virtually any ASN.1 type is supported, since transport is realized through object serialization. In the OSI stack implementation, we have defined all ASN.1 types supported by the underlying framework, i.e. simple types (boolean, all string types, integer,) and basic constructors (sequence, set, set of, sequence of, choice...). For all these types, Java classes are present, allowing the manipulation of ASN.1 values.

CMISoverJava provides two implementations: one is made over a full OSI stack and implemented over the OpenMaster Framework from BULL using the Java Native Interface. A second implementation of CMISoverJava is made using the Java Remote Method Invocation facility. Every application or agent developed in Java over this interface can use transparently any of these implementations.

7.2 MODERESJava MOF/GDMO ASN.1 translator

The *MODERESJava* environment exists since 1995 [22]. Initially used to allow the parsing of extended GDMO specifications, i.e. with formal behavior clauses, the toolkit was first extended to support the parsing and manipulation of GRM specifications [23] and was enhanced with various facilities such as pretty printers, semantics checkers and new information model loaders for MOF, SNMP SMI and TINA Q.GDMO/GRM [24-26]. The toolkit is freely distributed on the Web [20].

In the WBEM/OSI Q.Adapter environment, *MODERESJava* is used at two levels. A first one allows MOF specifications to be parsed and, based on our specification mapping algorithm, associated GDMO and GRM specifications to be generated (arrows *1* and *2* in figure 3). *MODERESJava* is also used to translate Generic GDMO definitions, ASN.1 types, generated GDMO specifications and GRM specification into Java classes (arrows *3* and *4*).

In the first release of the Q.Adapter, *MODERESJava* was also used to build CMIS Create Requests each time a MO instance or a MOF specification was found in the MOF file. These requests where used by a CMIS Manager to initialize the agent (arrow *5*). In the last release of the Q.Adapter, this function is obsolete in this way and is implemented directly in the agent, which performs automated discovery of CIM MO instances.

Later on, in the operational phase of the Q.Adapter, MODERES is used to parse incoming HMMP responses to extract MO related informations and translate them into ASN.1 values (arrow *5* on the right part of figure 3).

7.3 The Q.Adapter

The Q.Adapter as well as the whole software environment which realizes the mapping has been developed using the Java environment. It is designed as a stateful agent, which maintains mirrored information from the CIM agents it is associated to. The stateful information is in fact the GDMO mapping of the CIM classes defined in the CIM agents. Since these classes are supposed no to change very often in deployed agents, the corresponding `CIM_MetaMOC` class instances do not change either and thus represent the stateful part of the agent.
The Q.Adapter is a self-complete entity, which carries two main features:

- Automated discovery of a CIM agent MIB and instantiation of associated GDMO instances in its MIT, i.e. the Q.Adapter is able, given a CIM-agent address and translated MOF specifications, to dynamically discover its content (classes, namespaces, and MO instances) and build the corresponding part of the OSI MIB ;
- A traditional gateway approach, accepting CMIS operations on its MIB and propagating these requests to the associated CIM agent through HMMP and doing the opposite operation for incoming HMMP responses and/or events translated into either CMIS responses or notifications.

The main drawback of the current release of the Q.Adapter, is the fact that we do not take completely advantage of all the Java features, e.g. dynamic class loading. The first version of the Q.Adapter is quite basic and has to be recompiled every time one wants to add new GDMO associated Java classes. The dynamic and extensibility features of the Q.Adapter are under development and should be available within the next release. A second major drawback in the current implementation comes from the incomplete implementation of the Java HMMP API provided in the beta-releases of WBEM SDK. Now the problem is solved, since the API is no longer available and requires a native port of our calls.

8. Conclusion and future work

In this paper, we have presented the results of a project dealing with the integration of WBEM-based agents into an OSI Management Framework. Focusing on managing CIM-based agents through a TMN manager, we have shown that the integration in this way is feasible and quite easy to implement. The implementation was facilitated by the availability of both the *CMISoverJava* CMIS API as well as the *MODERESJava* toolkit, those tools providing well defined APIs for heterogeneous information model and CMIS requests as well as notification handling. For the mapping, we have chosen a *recast* mapping technique which seems more appropriate than a domain mapping because both approaches and available information models mainly address different resources and they are few intersections on modeled managed elements.
Several directions can be followed from this point. The first one consists in developing the other direction within the integration, i.e. GDMO/ASN.1 to MOF in a WBEM manager/OSI agent context and making the adaptor compatible with CIM 2.0 and the new COM communication interface. As already mentioned in the paper, some work is already ongoing on this subject in our group. Another approach may address

the integration WBEM in the same way it is done for OSI, SNMP and CORBAL-IDL in the JIDM approach but this is not yet addressed in the group.
Based on the developments we have made for building the Q.Adapter in a full Java environment also enables us to start experimenting the use of Java-based mobile agents in an OSI-based Management Environment. This is the main purpose within which we expect to use our environment, i.e. the TMN Java agent, *MODERESJava* and *CMISoverJava* communication API in the near future.

9. Acknowledgments

This work was partially funded as part of the ANTARES project between the RESEDAS research group and the OpenMaster division of BULL. The authors wish to thank all members of our research group who have contributed to parts of the prototype. Especially Eric Dillon, Serge Lassabe, Emmanuel Nataf and Didier Zhang who have been actively involved in the development of the CMISoverJava API used in this environment. The authors also wish to thank all contributors referenced in this paper for their contribution to the definition of integration approaches. All their proposals have contributed to facilitate our work.

10. References

[1] ITU-T, "Principles for a Telecommunications management network," ITU-T, International Standard M.3010, January 1996.
[2] M. Janders, "Web-based Management: Welcome to the revolution," *Data Communications International*, vol. 25, pp. 38-56, 1996.
[3] S. Todd, "HMMP Overview," Microsoft Corp., Experimental RFC May 1997.
[4] A. I. Rivière and M. Sibilla, "Management Information Models Integration: From Existing Approaches to New Unifying Guidelines," *JNSM*, vol. 6, pp. 333-356, 1998.
[5] S. Abeck, A. Clemm, and U. Hollberg, "Simply Open Network Management: An Approach for the Integration of SNMP into OSI Management Concepts," presented at ISINM'93, San Francisco, 1993.
[6] P. Kalyanasundaram and A. S. Sethi, "An Application Gateway Design for OSI-Internet Management," presented at ISINM'93, San Francisco, 1993.
[7] S. Mazumdar, S. Bradly, and D. W. Levine, "Design of Protocol Independent Management Agent to Support SNMP and CMIP Queries," presented at ISINM'93, San Francisco, 1993.
[8] NMF, "- Translation of Internet MIB-II to ISO/CCITT GDMO MIBs, Issue 1.0," Network Management Forum Forum027, 1996 1996.
[9] NMF, "ISO/CCITT to Internet Management Proxy, Issue 1.0," Network Management Forum Forum028, 1996.
[10] NMF, "- Translation of Internet MIBs to ISO/CCITT GDMO MIBs, Issue 1.0," Network Management Forum Forum026, 1996.
[11] A. Clemm, "SNMP and TL-1: Simply integrating management of legacy systems," presented at IM'99, San Diego, 1997.
[12] N. Soukouti and U. Hollberg, "Joint Inter Domain Management: CORBA, CMIP and SNMP," presented at IM'97, San Diego, 1997.

[13] Vertel, "Accessing TMN Through Web-based Entreprise Management," Vertel, White Paper 1997.
[14] ISO, "Structure of Management Information - Part 4: Guidelines for the Definition of Managed Objects," ISO, International Standard 10165-4, 1992 1992.
[15] ISO, "Structure of Management Information - Part 7: General Relationship Model," ISO, International Standard 10165-7, 1995.
[16] ISO, "Specification of Abstract Syntax Notation Number One (ASN.1)," ISO, International Standard 8824, 1990.
[17] DMTF, "Common Information Model (CIM) version 1.1," DMTF September 1997.
[18] CCITT, "Common Management Information Service Definition," ITU-T, International Standard 1991.
[19] ISO, "System Management - Part 16: Management knowledge management function," ISO, International Standard 10164-17, 1997.
[20] O. Festor, "The RESEDAS Free Java Management Software Homepage," INRIA, http://www.loria.fr/~festor/JAM/JAM.html 1997.
[21] T. R. Chatt, M. Curry, J. Seppä, and U. Hollberg, "TMN/C++ An object-oriented API for GDMO, CMIS, and ASN.1," presented at IM'97, San Diego, 1997.
[22] O. Festor, "MODE: a development tool for managed objects," presented at IM'95, Santa Barbara, 1995.
[23] E. Nataf, O. Festor, and A. L., "RelMan: a GRM-Based Relationship Manager," presented at IM'97, San Diego, 1997.
[24] O. Festor, "The GDMO and GRM Modules Semantic Checker of the MODERES Java Toolkit," INRIA, Technical Report RT-208, 1997.
[25] O. Festor, "MODE-PP HTML: A GDMO/GRM to HTML translator -Release 1.0- Reference Manual," INRIA, Technical Report RT-199, 1997.
[26] O. Festor, "The Managed Object Format specification parser of the MODERES Java Toolkit," INRIA, Technical Report RT-218, 1998.

Biography

Olivier Festor has been actively involved in network management research and development since 1991. Currently he is a senior researcher at the French National Institute for Research in Computer Science and Control (INRIA) where he leads a group on new technologies for network and service management in the TMN framework. He has published about a dozen papers in the domain of network and service management and holds a Ph.D. in computer science from the University Henri Poincaré in Nancy.
Paul Festor is probably the youngest author ever contributing to IM. Born on October 31st, 1998. Paul participated actively to the finalization of the paper in keeping his father awake late at night during the first weeks and letting him work on indecent hours to complete the contribution.
Laurent Andrey is a research engineer at BULL Corp. He is currently working in Nancy within the RESEDAS research group on both WBEM integration and the development of a Java CMIS API.
Nizar Ben Youssef is currently a Ph.D. student at the University Henri Poincaré Nancy I. His current interests are in management integration and the design of cooperative management architectures for managing active networks.