A Middleware Architecture for Distributed Systems Management ⋆

Jesús Salceda, Iván Díaz, Juan Touriño and Ramón Doallo

Department of Electronics and Systems, University of A Coruña, Campus de Elviña s/n, 15071 A Coruña, Spain

Abstract

This paper presents a middleware solution for global management of any kind of distributed system, such as networks of PCs/workstations, clusters or server farms. Our approach lies in an object-oriented software architecture that models all kind of management information using the Common Information Model (CIM) developed by the Distributed Management Task Force (DMTF). The classes and attributes obtained after the modeling process are mapped to a Lightweight Directory Access Protocol (LDAP) repository. This paper discusses the key features of our middleware that allows that any element (physical, logical, device, user or system) can be managed using a network-oriented and topology-independent approach. A representative example of management domain illustrates the procedures followed to build management applications using our middleware architecture.

Key words: Middleware, Management, Distributed Systems, CIM, LDAP

⋆ This work was funded by the Galician Government (Xunta de Galicia, Projects PGIDT01-PXI10501PR and PGIDIT02-PXI10502IF)

Email address: jsalceda@udc.es (Jesús Salceda).

1 Introduction

The management of distributed systems requires a set of tedious and time-consuming administration tasks (eg, OS installation, starting processes, package installation) to get the system up and running. Although some of these tasks can be performed in a semi-automatic way by using daemons for scheduling commands and shell scripts, all of them lack an overall management view. In this paper, we propose a middleware architecture for the management of distributed systems as a single system. Our research began in early 2001 with the aim of systematizing the management of the great variety of computing resources available at the School of Computer Science at the University of A Coruña. Our proposal uses an object-oriented model to represent management information as it provides support for abstraction and classification, inheritance, extensibility and the ability to depict any kind of element. Dependencies and relationships among different managed elements, like those existing among physical, logical, device, user or system, are also taken into account in our architecture. All management information is stored without redundancies into a common, network-accessible repository, where any managed element can query, update or retrieve the information stored in the repository through a common interface. Another key factor is the capability to systematically develop any kind of management task by using a common application program interface. Our procedure consists of the following steps: first, the problem domain and the management information are determined; next, this information is mapped to the network repository; finally, the set of tasks to manage each element are developed by application programmers or system administrators, allowing an easy integration into a more global application.

2 Related Work

Since networks of workstations have been widely deployed in academic and business environments, several well-known solutions (NIS, DNS ...) have been used to simplify administration tasks, but they focus on a limited scope. Any information infrastructure for global management must comply with several functional and architectural requirements. In [9] we can find a complete set of properties that are summarized in the following points:

- Global Repository: Configuration and administration data may be generated by different sources. However, global management needs a common distributed repository of information that allows decentralized maintenance through a lightweight network protocol and a common programming interface.
- Common Information and Naming Model: The management information model must be able to identify and model any element and the relationships among them in a similar way. Moreover, it should allow to straightforwardly incorporate any additional source of information in a near future.
- Scalability and cost: The management architecture should be able to scale up to a large number of managed elements, while performance, deployability, access and security cannot be affected by a given network topology.

Although the complex problem of distributed systems management has been a subject of intense research in the last years, the solutions we have evaluated, most of them focused on cluster management, do not comply with all the requirements described above. The Open Source Cluster Application Resources (OSCAR) [18], a cluster toolkit that includes several tools (SIS [8], LUI [17]

and C3 [10]) for managing clusters, uses a database only to store unrelated information (and thus unable to represent relationships) about initial installation and network configuration for each node. LCFG [1] automatically installs and manages the configuration of a large number of UNIX systems in a GRID infrastructure through a configuration language and a central repository of configuration specifications. However, this tool cannot be easily adapted to any network topology and uses a flat information model (key-value pairs) unable to model multiple dependencies. A related configuration tool is the NPACI Rocks Toolkit [13]. Although it uses a configuration infrastructure through well-defined inheritance properties, it focuses on complete OS installation, avoiding individual management of configuration files. It thus follows a heavyweight approach that limits deployment and performance when the number of managed systems increases. A management package was designed at Argonne National Lab to automate administration of the Chiba City cluster [16]. The software architecture focuses on the three-level hierarchical network of the cluster, and it is very difficult to extend to other network topologies. Moreover, it includes only management information that could conceivably change using single key-value pair records in a central database, which becomes a major bottleneck. Finally, a powerful cluster management approach has been implemented and deployed at Sandia National Labs [15]. It uses an object-oriented software architecture rooted by a Device Class, a persistent object storage to represent the physical and network topology of the cluster, and a set of layered management tools. However, this package has several drawbacks: the class hierarchy of this model focuses only on hardware devices, it is unable to represent complex relationships among elements, the naming model is not wide enough to refer unambiguously to any managed element, and there is not a distributed repository.

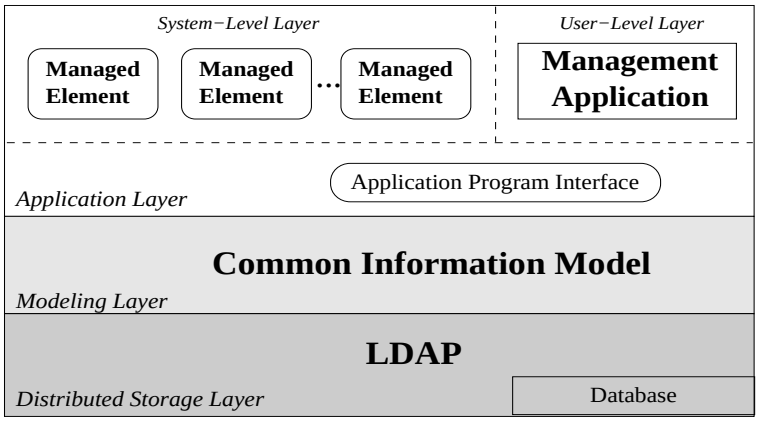

Fig. 1. Functional layers

3 Software Architecture Overview

We propose a software architecture to manage distributed systems that complies with the requirements enumerated in the previous section. Our proposal consists of the functional layers shown in Figure 1. First of all, we believe it is essential to make use of some kind of persistent object storage, like a database. This database stores and integrates all management information coming from several sources (ie, physical and logical elements). Moreover, it allows an easy integration of the upper layers needed to build management tools. Although the objects are stored in a database, we need a Distributed Storage Layer to overcome the drawbacks that databases have to provide a distributed repository in a transparent way. We have used the Lightweight Directory Access Protocol (LDAP) [12], a protocol for accessing directory services based on the client-server model, to build a network-oriented repository due to the capabilities for building a topology-independent distributed storage. LDAP directories define the information that can be stored in them through the schema [14]. In order to use LDAP directories as a management information repository, we must extend the standard schema to include all attribute

types and objectclasses necessary to represent this information, and this is the object of our Modeling Layer. The Distributed Management Task Force (DMTF) [5] has developed a basic information model known as Common Information Model (CIM) [2] that attempts to unify and extend the existing instrumentation and management standards using object-oriented design. CIM provides a mechanism for modeling information independently of any repository, although there is a set of guidelines [6] [7] to aid in mapping objects to directories that support the LDAP protocol and the X.500 model [11] (a successful implementation of this model over an LDAP directory is the DEN initiative [4]). CIM allows to build our management information model by providing us a common information and naming model. We have used the classes developed by DMTF for modeling the elements but, if there is not an appropriate class, we extend the model to include all the classes needed to manage the elements; in any case, we have to map all classes to the LDAP directory. The layered structure of our approach and the special characteristics of LDAP directories, in terms of distribution, allows to address scalability by enabling to build virtually any kind of aggregation of managed elements on a global distributed repository through delegation, or implementing high availability using superior capabilities like LDAPv3 replication [19].

The two upper level layers have to do with application development. Our software architecture considers two types of applications. On the one hand, System-Level applications are the set of procedures responsible for exchanging information between the managed element (eg, services, network interfaces) and the instances of these elements stored in the repository. Moreover, these applications modify the behavior of the managed element when the management information is updated in the repository. In general, the System-Level

Layer includes all kind of tasks we need to communicate the repository and the managed element. On the other hand, the User-Level Layer provides a single view for distributed systems management, as well as a high level of abstraction by hiding the special characteristics of the managed elements. For instance, at this level we can answer questions like "is the CERT Advisory CA-2002-20 installed in system A (RedHat 7.0), B (SuSE 8.1), C (Solaris 8) or D (AIX)?"

4 Modeling Layer

We have modeled a restricted management domain to illustrate the possibilities of our middleware. This domain tries to comprise a basic set of management tasks that system administrators need to perform on each managed element: (a) Network Interface properties: Access (MAC) and Network (IP) Layers, (b) Services started using System V-style `init` scripts (stored in `/etc/rc?.d` directories); and (c) Services started from the `inetd` superdaemon (stored in `/etc/inetd.conf` file). The first step to model our example domain is to build the information model to be mapped to the LDAP repository from DMTF-defined CIM classes. First, we must identify and check the CIM specifications to detect if the managed element is already modeled; otherwise, we have to extend the model to include it. Next, the CIM classes are mapped to the LDAP repository. The following description of our modeling is not intended to be comprehensive, but rather a proof of concept of the key features we have used.

CIM has several classes to model network interfaces. The *CIM_IPProtocolEndpoint* and *CIM_LANEndpoint* classes are used to store the information

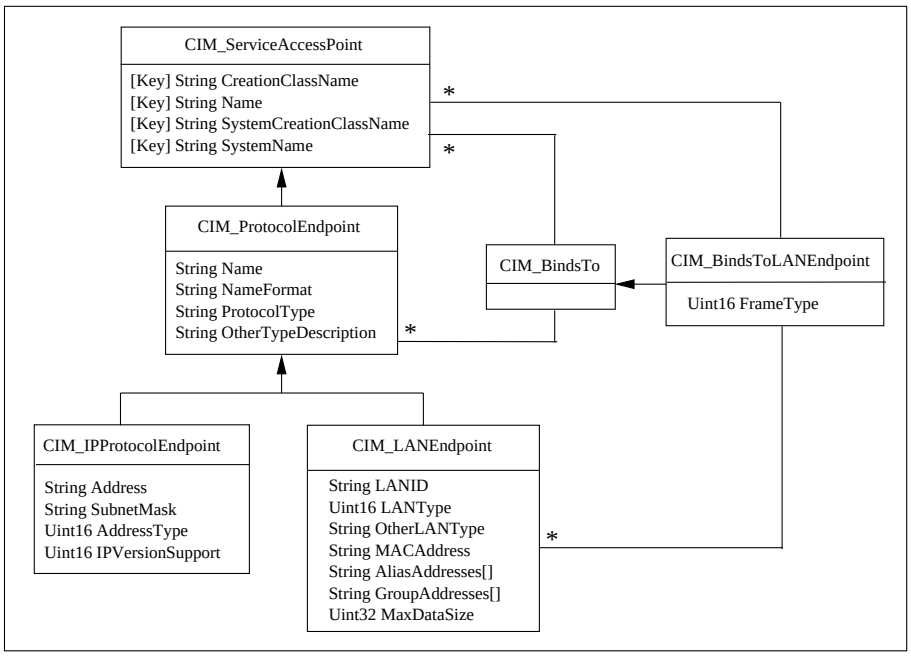

Fig. 2. CIM specification of network interfaces

about the communication points across which data can be sent to or received from the host at IP and MAC layers. The *CIM_BindsToLANEndpoint* class makes explicit the dependency relationship of the IP layer on the MAC layer. As can be seen in Figure 2, the *CIM_IPProtocolEndpoint* class contains the attributes we need to model the information about the IP layer of the interface: IP address, subnet mask, address type and IP version support. The *CIM_LANEndpoint* class includes the LAN identifier, LAN type, MAC address, alias MAC address, multicast address, and the maximum transfer unit of this interface. Both are subclasses of *CIM_ProtocolEndpoint*, and they inherit other properties from it, like the protocol type or the name. Moreover, both classes are related through the *CIM_BindsToLANEndpoint* association class to model the IP-MAC layer encapsulation. Next, we need to map these classes to the LDAP repository. As we can see in Figure 3, the attributes are mapped directly, but the classes need to be adapted. As these classes are concrete, our LDAP schema needs: (a) an abstract LDAP ob-

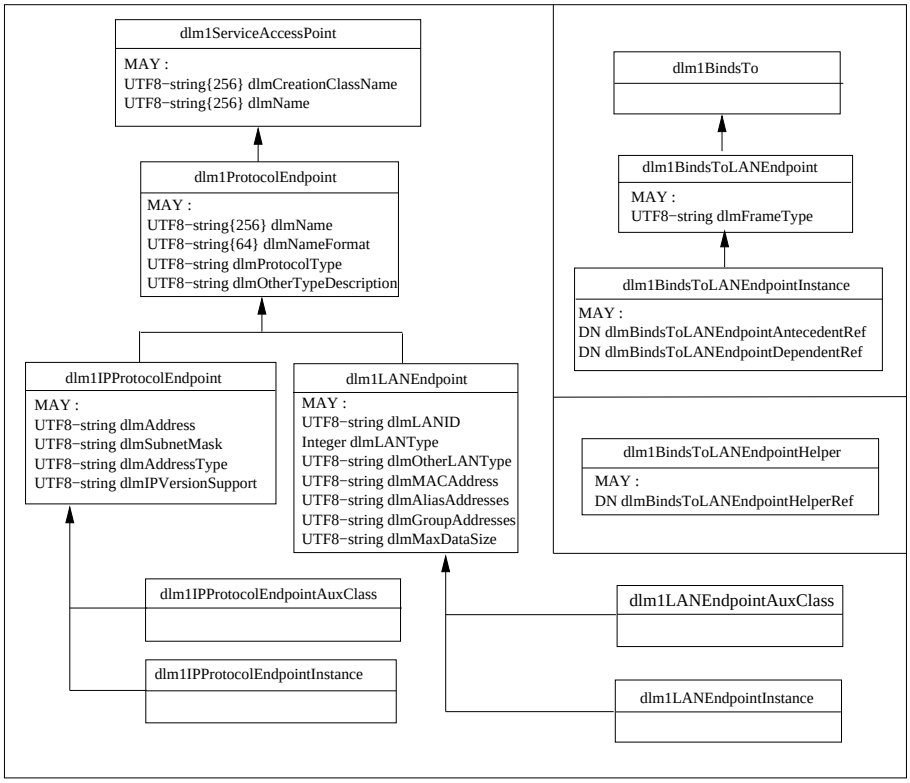

Fig. 3. LDAP objectclasses for network interfaces

jectclass, *dlm1IPProtocolEndpoint*, that inherits directly the attributes from the *CIM_IPProtocolEndpoint* class; (b) an auxiliary class, *dlm1IPProtocolEndpointAuxClass*, that allows to extend the original CIM model with additional attributes (if necessary); and (c) a directly instantiable structural class, *dlm1IPProtocolEndpointInstance*, used to create instances of IP interfaces. In a similar way, we have extended the LDAP schema to model MAC interfaces by using three objectclasses: *dlm1LANEndpoint*, *dlm1LANEndpointAuxClass* and *dlm1LANEndpointInstance*. The dependency between the IP and MAC interface layers needs a more complex set of LDAP objectclasses and attributes to map the *CIM_BindsToLANEndpoint* class. This dependency is a non-abstract association with a many-to-many cardinality. As depicted in Figure 3, we need a structural LDAP class, *dlm1BindsToLANEndpoint*, that contains the single property of the association, *dlmFrameType* (the framing method), as an op-

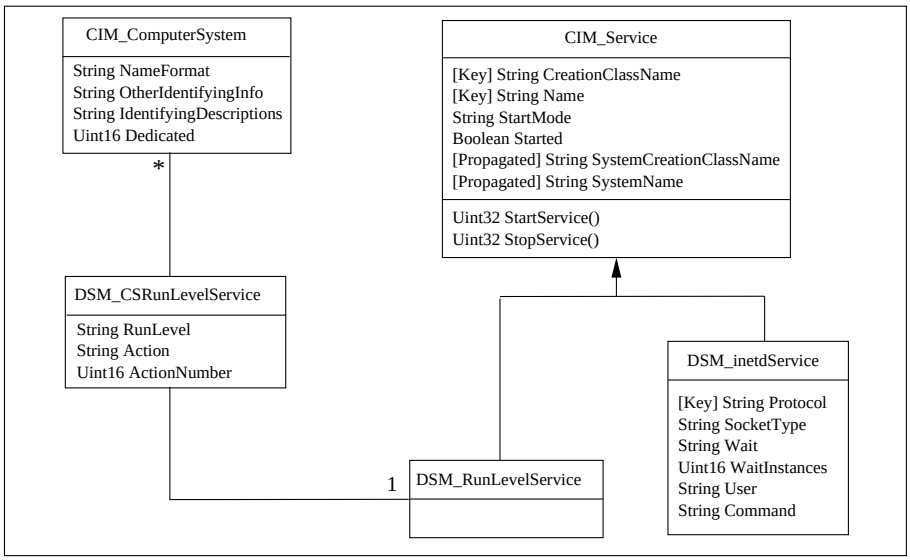

Fig. 4. CIM-extended classes of System V and `inetd` services

tional attribute. We also need a directly instantiable class, *dlm1BindsToLAN-EndpointInstance* (subclassed from the structural class), that includes the *dlmBindsToLANEndpointAntecedentRef/DependentRef* attributes, that is, the lower and upper layers of this association, respectively. Moreover, since this association is a separate object in the directory, the auxiliary class *dlm1Binds-ToLANEndpointHelper* must be attached to each instance of object belonging to this association. This helper class includes a single optional attribute, *dlmBindsToLANEndpointHelperRef*, that points to the particular instance of the association in which the LAN interface participates.

As explained in Section 3, a key characteristic of CIM is the capability to include additional schemas to represent management information. This is the case with System V-style `init` scripts and services started from the `inetd` superdaemon. Unlike network interfaces, it was necessary to extend the model to include management information about these services. As shown in Figure 4, our starting point is the *CIM_Service* abstract class from the CIM Core Model [3] that contains the information necessary to represent the function-

ality provided by a device or software element. We have introduced three new classes: *DSM_RunLevelService* and *DSM_CSRunLevelService* to model `init` scripts executed on a computer system (*CIM_ComputerSystem* class) for a certain runlevel; and *DSM_inetdService* to model `inetd` services configured by the `/etc/inetd.conf` file (the *DSM* prefix stands for Distributed Systems Management). The LDAP mapping of all these classes is similar to that followed by network interfaces.

The next step in our Modeling Layer is to provide an unambiguous identification of the managed elements. As the common naming model we have used is based on the LDAP naming model [14], we need a method to adapt CIM to the LDAP namespace. The basic unit of this namespace is the entry, an instance of one or more objectclasses. Each entry is uniquely identified by a Relative Distinguished Name (RDN) built with at least one of its attributes; in our case, the attribute *orderedCIMKeys*, constructed by ordering the CIM keys of the entry. The concatenation of the RDNs of all parent nodes in the Directory Information Tree (DIT) builds the Distinguished Name (DN) of the entry to identify the concrete element. The DIT of the LDAP namespace allows to model any kind of relationship, such as the CIM class hierarchy; in this case, the propagated keys may be omitted in the LDAP namespace. The following example illustrates the common naming model for an Ethernet interface `eth0` in a host named `yogsototh`. The modeling of the interface results in a set of LDAP entries to represent elements and associations. As shown in Figure 5, we have used three LDAP entries to store the information about `eth0`: IP Layer (entry numbered as 1), MAC Layer (entry 2) and IP-MAC layer encapsulation (entry 3). Several key aspects can be extracted from them:

- The RDN of the interface `eth0` in the IP and MAC layers (entries 1 and

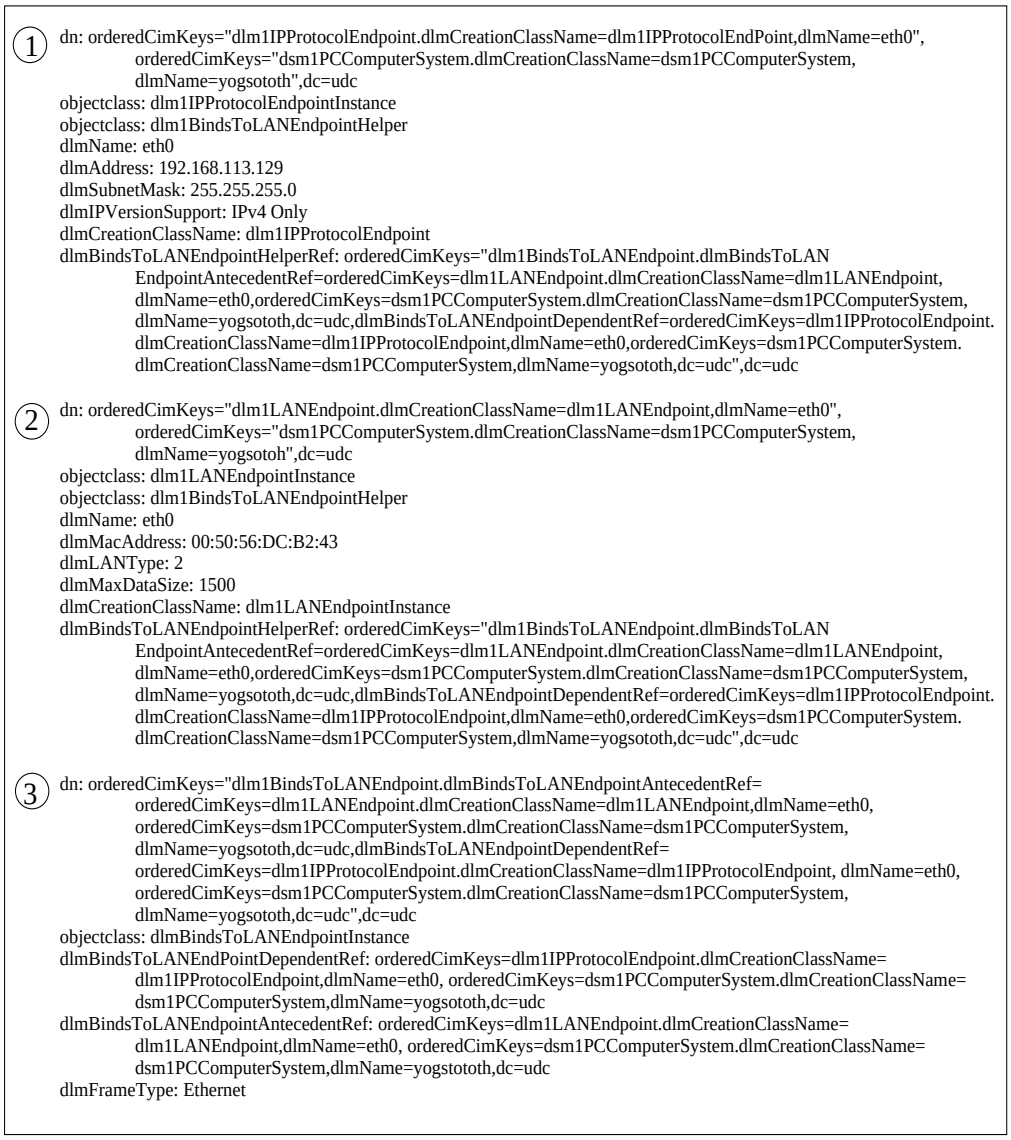

Fig. 5. LDAP network interface entries

2) is built using its key attributes: *dlmCreationClassName* and *dlmName*, both inherited from *dlm1ServiceAccessPoint* objectclass (see Figure 3).

- The DN of entries 1 and 2 uses the DIT to represent the weak association between the interface and the computer system (the PC host). The root of these entries is the internet domain where both elements are included.
- The properties of the interface are represented by values of the attributes using a consistent naming model.
- The key attributes of the association between the two layers of the interface

`eth0` (entry 3) are the DNs of the antecedent and dependent instances.

- Each `eth0` entry (entries 1 and 2) has a field *dlm1BindsToLANEndpoint-HelperRef* with the DN of the association where this entry is included.

5 Application Layer

Once we have the basis of our architecture, it is necessary to develop a set of programs to address the two basic issues of building management applications in our approach: the bidirectional information transfer between managed elements and LDAP repository (System-Level Application), and the specific management application (User-Level Application).

The System-Level Layer consists of a set of procedures written in Perl. They transfer configuration information, both from files and system administration commands located in the managed system, to the LDAP repository. In the other way, there are procedures that maintain the local system information consistent with the information updates performed in the directory. So, this layer converts the raw information stored in each local system into an object-oriented format (objectclasses and attributes) suitable for the Modeling Layer. Although the nature of the information stored in each system is diverse it comes, in general, from at least one of the following sources: configuration files, output from command line executions, and directory structures. The most important way of maintaining management information are configuration files stored in local filesystems. As these plain files have little structure (or none), we cannot use generic parsers to extract the information they store and thus, they must be approached in an individual manner. In these files the line is usually the basic information unit, so that they must be processed

```
sub registerService {
$Name = shift; SocketType = shift;
$Protocol = shift; $Wait = shift;
$User = shift; $Command = shift;
$ldap = Net::LDAP->new($machine) or die "$@";
auth();

$ock="dsm1PCComputerSystem.dlmCreationClassName=dsm1PCComputerSystem,
dlmName=".$fqdnHostName;
$PCdn="orderedCimKeys=\"",$ock."\",dc=udc";

$dn="orderedCimKeys=\"dsm1inetdService.dlmCreationClassName=
dsm1inetdService,dlmName=".$Name.",dlmProtocol=.$Protocol."\",".$PCdn;
$mesg = $ldap->add ($dn,
        attrs => [ objectClass => 'dsm1inetdService',
                   dlmCreationClassName => 'dsm1inetdService',
                   dlmName => $Name,
                   dsmSocketType => $SocketType,
                   dsmProtocol => $Protocol,
                   dsmWait => $Wait;
                   dsmUser => $User;
                   dsmCommand => $Command ] );
        $mesg->code && die $mesg->error;
        DSM::Util::makeAuxAssociation ($ldap,$PCdn,$dn,
             'dlmHostedServiceAntecedentRef',
             'dlmHostedServiceDependentRef',
             'dlm1HostedService');
}
```

Fig. 6. Perl code of *registerService* function

line by line to capture the configuration information from the corresponding fields. Once the line information is stored in variables, we need to create the corresponding LDAP entries. The function *registerService* shown in Figure 6 is an example of this process for the `/etc/inetd.conf` file. This function opens an LDAP object (see fifth line of the code) and builds its DN through the *orderedCimKeys* attribute (see Section 4). Next, the function adds the entry using the `$ldap->add` method and, finally, the associations for this object are defined. The second source to extract configuration information is the output of configuration commands and programs. This is a very useful way to get data from the OS kernel, or to gather information coming from several different sources (like the `ps` command that collects username, PID, command name ..., or the `ifconfig` command for network interface configuration). The third source is the directory structure. A representative example is a service started by `init` scripts. The procedure followed to capture information from these two sources is very similar to that used for configuration files.

Regarding the other way of information transfer (repository-to-element), the procedures that transfer any change performed in the LDAP directory to the managed elements follow a two-phase approach: (a) notify to the managed elements, and (b) do the appropriate actions in them. The notification, depending on the managed element and the kind of change, can be made by triggering or polling. In our domain, the kind of actions to be taken in the managed element are, for instance: update of configuration information (files, directories or commands), start/stop of individual services or set of services, reload or restart of daemons, or start/stop of groups of systems.

The objective of the middleware described so far is to provide a common infrastructure to develop applications for managing distributed systems as a single system. In order to complete the cycle we have developed an example of user-level application in Java with the following functionalities: (a) Selection of managed elements: interfaces, services; (b) Selection of hosts (individual or groups) to be managed; (c) Network interface management: query information, start/stop network interfaces, check status; (d) Service management (System V and `inetd`): query information, start/stop services, check status, change runlevel (including system shutdown). Although not described in the paper, ACL (Access Control List) management is also included: query information, add/delete/update access control entries in `hosts.allow/deny` files. Regarding implementation issues, the application uses a tree abstraction through the `JTreeModel` class that allows to construct a tree dynamically from the directory information, and represent it in terms of real world objects, like network interfaces or services. The application provides a view (reasonably close to the CIM view) that conceals details like for instance the internals of the various types of services, while the user is only concerned with their management.

6 Conclusions and Future Work

We have proposed a middleware architecture to manage distributed systems that follows a multilayer approach: (a) a Distributed Storage Layer, a network-oriented LDAP-based storage where persistent objects that represent management information are stored; (b) a Modeling Layer based on the extensible CIM to model the information, including the relationships among the managed elements; and (c) an Application Layer where the low-level transfers between the directory and the managed elements (System-Level Layer), and customized high-level management applications (User-Level Layer) can be developed using a common application program interface. The layered architecture hides the different way operating systems manage similar elements through the use of a common information and naming model. A representative management domain was used to illustrate the main procedures of our approach: mapping of CIM objects to an LDAP repository, extension of the model to include new elements to be managed, development of reusable functions to collect management information from different sources, and management tasks that involve dependencies among elements (eg, IP address modification, start/stop services).

Future work focuses on extending the management domain by including new elements (and their corresponding relationships) such as hardware devices, file systems and volumes, software packages, system processes, log events, users/groups information for authentication purposes, and network security. We are also considering the use of XML as a common intermediate language in all middleware layers (CIM schemas, LDAP entries, system-level/user-level applications) to improve the processing of the management information.

References

[1] P. Anderson and A. Scobie. LCFG: The Next Generation. UKUUG LISA/Winter Conference. London, UK, February 2002.

[2] DMTF. Common Information Model (CIM) Standards. http://www.dmtf.org/standards/standard_cim.php

[3] DMTF. Core MOF Specification 2.6. June 2002. http://www.dmtf.org/standards/documents/CIM/CIM_Schema26/CIM_Core26.mof

[4] DMTF. Directory Enabled Network (DEN) Initiative. http://www.dmtf.org/standards/standard_den.php

[5] DMTF. Distributed Management Task Force Inc. http://www.dmtf.org

[6] DMTF. Guidelines for CIM-to-LDAP Directory Mappings. May 2000. http://www.dmtf.org/standards/documents/DEN/DSP0100.pdf

[7] DMTF. LDAP Schema for the CIM v2.4 Core Information Model v1.0. May 2002. http://www.dmtf.org/standards/documents/DEN/DSP0117.pdf

[8] B. Finley, S. Dague, M. Chase-Salerno and D. Frazier. System Installation Suite (SIS). http://sisuite.org

[9] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. 6th IEEE International Symposium on High Performance Distributed Computing, pages 365-375. Portland, OR, August 1997.

[10] A. Geist, M. Brim, B. Leuthke, S. Scott and T. Naughton. Cluster Command and Control (C3) Project. Oak Ridge National Laboratory. http://www.csm.ornl.gov/torc/c3

[11] S. Heker, J. Reynolds and C. Weider. RFC 1309 - Technical Overview of Directory Services using the X.500 Protocol. http://www.ietf.org/rfc/rfc1309.txt

[12] J. Hodges and R. Morgan. RFC 3377 - Lightweight Directory Access Protocol (v3): Technical Specification. http://www.ietf.org/rfc/rfc3377.txt

[13] M.J. Katz, P.M. Papadopoulos and G. Bruno. Leveraging Standard Core Technologies to Programmatically Build Linux Cluster Appliances. 4th IEEE International Conference on Cluster Computing, pages 47-53. Chicago, IL, September 2002.

[14] K.D. Keilenga. Internet Draft - LDAP: Directory Information Models. http://www.ietf.org/internet-drafts/draft-ietf-ldapbis-models-06.txt

[15] J.H. Laros, L. Ward, N.W. Dauchy, R. Brightwell, T. Hudson and R. Klundt. An Extensible, Portable, Scalable Cluster Management Software Architecture. 4th IEEE International Conference on Cluster Computing, pages 287-295. Chicago, IL, September 2002.

[16] J.P. Navarro, R. Evard, D. Nurmi and N. Desai. Scalable Cluster Administration - Chiba City I Approach and Lessons Learned. 4th IEEE International Conference on Cluster Computing, pages 215-221. Chicago, IL, September 2002.

[17] Open Cluster Group. LUI: Linux Utility for Cluster Installation. http://oss.software.ibm.com/developerworks/projects/lui

[18] J. Squyres, S. Scott, M. Chase-Salerno, S. Dague and N. Gorsuch (Open Cluster Group). Open Source Cluster Application Resources (OSCAR). http://oscar.sourceforge.net

[19] E. Stokes, R. Weiser, R. Moats and R. Huber. RFC 3384 - Lightweight Directory Access Protocol (version 3) Replication Requirements. http://www.ietf.org/rfc/rfc3384.txt