Development of Management Entities for Heterogeneous Scenarios

Rui Pedro Lopes (rlopes@ipb.pt)
Instituto Politécnico de Bragança, Escola Superior de Tecnologia e de Gestão, Bragança, Portugal

José Luís Oliveira (jlo@det.ua.pt)
Universidade de Aveiro, Departamento de Electrónica e de Telecomunicações, Aveiro, Portugal

Abstract

Managed agents are essential in communication networks. However, the development of agents is an expensive and a complex task. Although there are tools available which intend to simplify this process by generating automatic code based on the management information definition (MIB files), these are usually complicated to use and are generally coupled with a specific protocol stack.

This paper describes an API for managed agent development which provides multiprotocol capabilities. Using the same code, the resulting agent can be managed by SNMP, web browsers, wap browsers, CORBA or any other access method either simultaneously or individually. The communication engines are provided as *plug-ins* that can be installed in runtime.

I. Introduction

Research communities frequently tend to consider the development as the less noble phase of the research timetable. In this scenario, software development is largely seen as a simple engineering task and typically do not capture too much attention. However, the synergy between development and modelling frequently set up interesting issues that are underestimated in the analysis and in the architectural phases.

SNMP products have been around for a decade and have conquered the market for network and system management. While the maturity of products and the increasing acceptance of this protocol have spread the idea that the research on this mater is more or less stable, the development of solid products from scratch shows that this is in fact a misconception.

The early motivations for our study were centred on the topic "distribution management and SNMP". This goal led us to the evaluation of documents and proposals for several IETF charters (like the Distributed Management charter) [1] and to propose new solutions related to the distribution of management tasks. On this trail, and since most of the proposals did not presented yet any implementation, we decided to develop several MIB modules in order to have a solid knowledge of what was under assessment. Examples of this work are the Schedule, Event and Expression MIB [2] modules of DISMAN and the MAF-MIB defined to deal with mobile agents under SNMP [3].

While some results from this research have been written and presented, we realize that a main piece of the work was taking little attention. In fact it was mainly a development feature – although crucial within the overall architecture. This paper presents an approach to the development of management agents for heterogeneous scenarios, i.e., an API that allows a single development strategy to build, for instance, SNMP agents, CORBA agents and HTTP agents.

II. Agent API

Usually, the development of a managed agent is ruled by the following procedure (using SNMP as a reference):

1. Define or retrieve the MIB definition in SMI.
2. Generate the source code (mostly C, C++ or Java) correspondent to the MIB module through a specific MIB compiler.
3. Update the generated source code with the agent functionality (programming).
4. Compile, test and deploy the SNMP agent.

At the time of the initial development, the tools which generate Java source code were sparse and many required commercial licenses. It was not possible to find a reasonable public domain tool and the available commercial tools were too expensive. Moreover, typically, its functionality was not clearly defined.

The available tools generate large and monolithic code (i.e., a single source file for each SMI file, sometimes mixing similar but unrelated management concepts) which results in increasing difficulties for the programmer. On the other hand, the tools are usually dependent on the SNMP stack. The use of other protocols or access methods is not easy without gateways, proxies or explicit programming.

To cope with these difficulties we decided that a specific, full featured API for the development of management agents was required – the AgentAPI. Replacing the MIB compiler approach, the programmer should be able to develop an agent by extending classes and invoking methods on a software layer which provides all the common and low level aspects, such as protocol message processing, agent extension, managed object identification and others the same strategy is already commonly used in GUI and client/server APIs (HTTP servers, database access, and Peer to Peer applications).

A. Design decisions

The fundamental design goal was the creating of a unique point of access that transparently adapts requests and responses to and from the selected transfer protocol. Mapping specific management concepts such as 'agent', 'agent object', 'MIB table' and others as classes in an object oriented language reduces the gap between theoretical concepts and practical development.

The agents built around the AgentAPI should be accessible by SNMP in all the three versions: SNMPv1, SNMPv2c and SNMPv3. Because we did not want to develop the SNMP

stack (there are several high quality implementations available either commercially and in public domain such as JoeSNMP, Westhawk, ModularSNMP or AdventNet SNMP stacks), we provide an interface between the AgentAPI core functionality and the SNMP stack, thus achieving version independence.

We also required that we could build sub-agents through the AgentAPI, connected to master agents by an extensibility protocol such as AgentX [4]. This choice would allow integrating new agents into existing SNMP agents.

Continuing the list of requirements, we also wanted direct agent access through regular Web browsers thus allowing the manager to retrieve and modify management information anytime, anywhere. Moreover, some minor changes allowed us to extend the access method to other devices, such as WAP browsers. This approach helps solving, on one hand, the restricted access to network management console by diversifying the access along a multitude of terminals, including handheld devices. On the other hand, we also aim at reducing the dependence of management operations (SNMP requests) on a central management system through the adoption of distributed management mechanisms.

This multitude of access methods requires a design able to support several others protocols or access methods, such as Java RMI (Remote Method Invocation) and CORBA.

Regardless of the access methods, the AgentAPI core should be able to manage all the common agent mechanisms, such as object management (ordering, creating, removing and activating managed objects) as well as allowing object persistency.

B. Architecture

The model is structured in three main parts: the management object handling, the communication mechanism and the extensions (Figure 1).

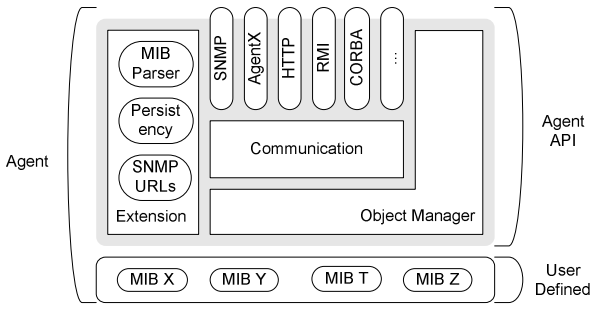

Figure 1: AgentAPI high level architecture.

The object manager contains the instrumentation part of the agent, which follows the SMI description of required MIBs. This information store defines how protocol commands are mapped to platform operations, by modifying or retrieving working parameters. Moreover, this layer deals with other intrinsic aspects such as dynamic creation of table rows, the ordering of management objects for GetNext handling, table indexing and others.

The communication module permits to use a broad set of protocols. For instance, if web-oriented interfaces are required the system can benefit from using the HTTP protocol. To maintain compatibility with traditional network management stations any SNMP version or AgentX can be used. The communication engines are created according to the user requirements. For example, an agent using exclusively SNMP will only need the SNMP stack – a normal SNMP agent. Other agents may require other protocols, such as HTTP, SSL, RMI or CORBA.

Regardless of the intrinsic operational differences and of the access protocol, agents may share some common features such as persistency, MIB parsing and command identification and processing. These operations are grouped into a transversal block, or extensions, thus improving code reusability and organization.

C. Object Manager

The above mentioned modules result in a set of classes which will be specialized to define specific agent behaviour. An agent, in the context of the AgentAPI, is a specialization of the core class (`AbstractAgent`) which uses a binary tree structure to store references to agent objects (`AgentObject`). It also maps these references to OIDs (Object Identifiers) which provides the correspondence between managed objects and memory objects. The tree structure has higher efficiency for "walk" operations than arrays or hash tables due to the OID ordering scheme.

The managed objects may be specializations of three kinds of classes: simple objects, node objects or MIB tables. Every object inherits the SNMP related set of operations from `AgentObject` and introduces a new set, therefore providing the specialization required by the MIB module.

Table objects are not intended to be derived (although permitted). To drive the table behaviour a `TableModel` may be used. It defines the columns type and number, validates the information and generates a repository object for each row of data (`TableProvider`).

D. Communication engines

The communication module adapts different communication engines to provide support for different protocols. Messages may flow towards the agent, such as the commands issued by management applications, or in the opposite direction, such as notifications.

The interface `MessageListener` is the responsible for defining the methods used in the communication between protocol engines and the agent. Any class which implements this interface can be used as a communication engine. The agent also implements the same interface, allowing bi-directional communication.

The engines for SNMP and AgentX makes the agents developed with the AgentAPI look like any other SNMP agent. However, other engines, such as HTTP provides different access methods which can be used in different situations. With a careful data transformation the HTTP client

can be a regular Web browser or a WAP browser through a gateway usually residing at the Internet Service Provider (an online demo is available in http://nms.estig.ipb.pt/see/disman to browse the agent through a common Web browser, and in http://nms.estig.ipb.pt/see/disman?wml=true for accesses through WAP terminals).

The transformation implies that the structure of management information (MIB definition) inherent to the agent must be somehow transformed to HTML, WML or others. Advances in web technologies suggest mechanisms to transform XML code into custom code, including HTML, text or WML by XSL transformation [5]. To be able to use this mechanism it is necessary to describe the structure of agent information in XML. Moreover, past IRTF (Internet Research Task Force) work under the NMRG (Network Management Research Group) focused on defining a Document Type Definition (DTD) to allow XML parsing applications to read or edit original SMI definitions [6].

We have built a communication module where the XML definition is complemented with a XSLT post-processor that dynamically generates management views in a format that is the best suited for the client's interface (Figure 2). Naturally this strategy implies larger agents. However, the price is acceptable when dealing with complex agents (collector or distributed manager for instance) where the overcharge is negligible and when it is important to have ubiquitous access to agents.

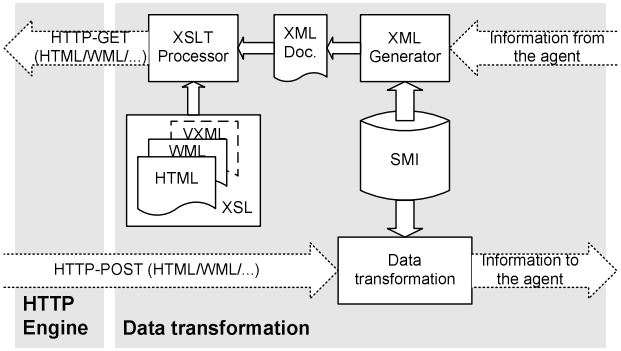

Figure 2: XSLT processor on SMI information architecture.

The communication module takes two sources as input: SMI files for the agent MIB structure and the agent information for the values. This joined information is converted to XML by the XML Generator and is then forwarded to the XSLT Processor. The XSL sheets provide the guidelines for transforming the common XML input to different output documents, such as HTML, WML or VXML. The result is then sent to the embedded HTTP server.

The XML generator builds a document based on SMI definition and augments the structure with the data from the local SNMP agent.

When the user accesses the HTTP engine, a login page shows up. This provides a minimum security level through user authentication. After being successfully authenticated, the user can monitor and control (through HTML or WML pages) the state of the agent. More security may be achieved by using HTTP over SSL.

E. Extensions

The AgentAPI is complemented with a very complete and full featured SMIv2 parser which allows converting MIB description files into Java objects by including very simple code (further programming examples are available at the project web page [7]):

```
MibModule module = MibOps.load("SNMPv2-MIB");
```

To extract the information, the procedure is also straightforward:

```
MibIdentity identity = module.getIdentity();
System.out.println(identity.getOrganization());
```

The extensions module has also a persistency mechanism based on XML, to provide features such as recording and retrieving the agent operation state [8], and a parser to SNMP URLs [9].

III. USAGE SCENARIOS

The actual trend in the telecommunications industry is towards integration. We see today crescent ubiquitous wireless access in corporation domain and the implementation of political initiatives like the Portuguese *Campus Virtual* program [10].

On the other hand, technologies such as uniform messaging, IP telephony and ad-hoc networks are now part of our daily life, with the implicit diversity, heterogeneity and user mobility.

The management of such networks becomes increasingly important and network management systems must cope with its dynamism.

A. Test bed

We have validated and used the AgentAPI in some research projects related to the DISMAN work and management distribution by developing some rather complex MIBs, namely the Schedule, Expression and initial work with the Event MIB modules. Moreover, we have used it to implement a custom made MIB to manage mobile agents throw SNMP and MAF [3], which maps MAF interfaces [11] to SMI thus allowing the definition of a gateway between SNMP and any compliant mobile agents' platforms. The agent instrumentation is performed through CORBA method invocations which interact with the agents' and platforms' life cycle. It also provides searching capabilities to locate resources in given regions.

The AgentAPI resulted in a valuable tool which provides common agent mechanisms such as object ordering, command processing and multi-protocol access.

B. Availability

Test bed results are further refined by making the API available to the Internet community. The AgentAPI is freely available under the GNU Public License [12] including all the tools and source code. In the year of 2002 we registered over 1000 downloads which revels a good community

acceptance and forecasts reasonable usage experience. We already received strong encouragement notes by users all over the world which reported positive usage experiences.

IV. Conclusions

The development of management agents is some times a rather complex task. The community has already proposed and deployed several tools to help the developer by generating code based on the definition of the agent's managed objects. These tools are usually expensive and generate large and redundant code and are also very tied to the SNMP model.

To cope with these difficulties we have developed an open source and extensible API, gathering all the common agent procedures. This API is extended to define specific agent behaviour as defined in the MIB module. Moreover, it allows using several simultaneous communication mechanisms, allowing direct access to the agent through SNMP, RMI, CORBA, HTTP, WAP, or AgentX.

V. References

[1] DISMAN, Distributed Management Charter, (http://www.ietf.org/html.charters/disman-charter.html).

[2] Lopes, R., Oliveira, J., "Delegation of Expressions for Distributed SNMP Information Processing", *accepted for publication in proc. of the 8th IFIP/IEEE International Symposium on Integrated Network Management – IM2003*, Colorado Springs, Colorado, USA, March 2003.

[3] Lopes, R., Oliveira, J., "SNMP Management of MASIF Platforms", *proc. IFIP/IEEE International Symposium on Integrated Management 2001 – IM2001*, May 2001, Seattle, USA.

[4] Daniele, M., Wijnen, B., Ellison, M., Francisco, D., "Agent Extensibility (AgentX) Protocol Version 1", *Internet Request for Comments 2741*, January 2000.

[5] XSL Transformations (XSLT), W3C Recommendation 16 November 1999 (http://www.w3.org/TR/xslt).

[6] Schoenwaelder, J., Strauss, F., "Using XML to Exchange SMI Definitions", *Internet Draft draft-irtf-nmrg-smi-xml-00.txt*, June 2000.

[7] AgentAPI (http://nms.estig.ipb.pt/).

[8] Lopes, R., Oliveira, J., "A new mechanism for distributed managers persistence", *proc. of the 3rd International Conference on Enterprise Information Systems – ICEIS2001*, Setubal, Portugal, July 7-10, 2001.

[9] Lopes, R., Oliveira, J., "A uniform resource identifier scheme for SNMP", *proc. of the 2002 IEEE Workshop on IP Operations and Management - IPOM2002*, Dallas, Texas, USA, October 2002.

[10] Amaro, J., Lopes, R., "A wireless MAN in Bragança - Digital City", *proc. of the 3rd Conference on Telecommunications – ConfTele2001*, Figueira da Foz, Portugal, April 2001.

[11] "Mobile Agent Facility Specification", *Object Management Group*, (ftp://ftp.omg.org/pub/docs/formal/00-01-02.pdf).

[12] GNU Public License (http://www.gnu.org).