Proceedings of the *Third International Enterprise Distributed Object Computing Conference: EDOC'99*, Mannheim, Germany, September 1999

Dynamic Management of Internet Telephony Servers: A Case Study based on JavaBeans and JDMK

Alexander Keller*
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598, USA
Email: alexk@us.ibm.com

Helmut Reiser
University of Munich, Dept. of CS
Oettingenstr. 67
D-80538 Munich, Germany
Email: reiser@informatik.uni-muenchen.de

Abstract

It is widely recognized that the scalable and efficient management of large, heterogeneous information technology infrastructures requires middleware that allows the extension of managed resources with new functionality at runtime. A recent and promising approach is the Java Dynamic Management Kit (JDMK) that relies on the JavaBeans component technology. Based on our experiences with a prototype implementation that provides an adaptable management instrumentation for an Internet Telephony Server, this paper analyses the suitability of JDMK for building scalable and flexible management systems. The results of our work allow us to evaluate JDMK regarding its suitability for the management of large-scale enterprise networks.

Keywords:
JavaBeans, JDMK, JMX, Internet Telephony Server, Distributed Management, Management by Delegation

1 Introduction

The ever-increasing deployment of electronic commerce solutions and the computerized interworking between manufacturers, vendors and customers requires fault-tolerant and scalable network infrastructures. Integrated management of corporate networks, systems and applications helps service providers to maintain a high level of quality by providing mechanisms able to cope with large and highly complex environments. Today, the dynamic distribution of management services to extensible agents residing on the managed resources is considered as the most promising approach for addressing the challenge of scalable, integrated management. Although significant progress has been made recently, the application of "traditional" (i.e., protocol-based) management architectures like the OSI/TMN or Internet (SNMP-based) frameworks to networked e-business applications is still at an early stage. This is – among others – due to the fact that management software based on these frameworks being able to transfer lightweight management components between management entities is not yet available on the marketplace.

On the other hand, the success of principles from distributed systems engineering motivates the development of modular management systems built from off-the-shelf software components. This leads to the demand for technologies that shield developers from the heterogeneity of the underlying systems. In this context, object-oriented implementation frameworks such as CORBA and JavaBeans are gaining increasing importance for the integrated management of networks, systems and applications.

The *Java Dynamic Management Kit (JDMK)* [19] by Sun Microsystems, Inc. aims to achieve the goal of scalable, distributed management by allowing the development of flexible management components that can be modified at runtime and then loaded on the managed resources. Based on our experiences with building a JDMK-based prototype for managing Internet telephony servers, we were able to evaluate to which degree JDMK fulfills the above requirements.

Internet telephony servers have a high potential for reducing an enterprise's communication costs by acting as a gateway between private branch exchanges (PBX) and the Internet. On the one hand, they have very high requirements with respect to availability and reliability but are, on the other hand, also deployed in increasing numbers. Consequently, Internet telephony servers have severe scalability constraints and make their management a challenge. The work described in this paper has been carried out in a joint cooperation project with Siemens AG, Munich, Germany [9].

The paper is structured as follows: In the next section we will describe the architecture of the Siemens Internet telephony server and its current management instrumentation. Based on an analysis of management requirements for Internet telephony servers, we discuss the shortcomings of

*This work was carried out and completed while the author was working at the Munich University of Technology.

the current approach and the steps to enhance their manageability in large IT infrastructures. While section 3 gives an overview of the Java Dynamic Management Kit architecture, its management services and development environment, section 4 describes how we applied JDMK to the problem domain of Internet telephony servers. Section 5 summarizes our experiences and evaluates whether JDMK can serve as an infrastructure for scalable, reliable and dynamic management solutions. We will discuss why some of the criteria defined in section 2 are not met by the current version of the toolkit.

In June 1999, the new *Java Management Extensions (JMX)* specification has been released for public review; it is intended as a replacement for the *Java Management API (JMAPI)* and relies to a large extent on JDMK. We will therefore contrast JMX with JDMK in section 6 and discuss to which extent JMX alleviates the shortcomings of JDMK that we identify in section 5. Section 7 concludes the paper and presents issues for further research.

2 Internet Telephony Servers: Architecture and Management

In order to reduce the costs for buying, operating and maintaining communication services and devices, the integration of voice and data is gathering increased attention. It is possible to use the high capacity of already established broadband data networks for voice transmission, thus overcoming the need to rely on public carriers for long-distance telephony: Enterprises now have the opportunity to locally transform outgoing calls to remote sites into data streams that are then transmitted through the Internet. At the remote site, incoming streams are converted back into voice and delivered to the addressee, thus bypassing public switched telephone networks (PSTN). The potential of multi-million dollar cost savings per year makes the integration of voice and data particularly attractive for global enterprises with large corporate networks.

To achieve this, gateways between private branch exchanges and the Internet have to be installed and deployed at multiple locations of an enterprise. These gateways are termed **Internet telephony servers**. We will now describe the system which our implementation is based on.

2.1 Siemens Telephony Internet Server

By means of different gateways, the Siemens ISDN private branch exchange (PBX) Hicom 300 is able to connect telephone calls over ATM or over Internet protocols (TCP/IP) [16]. The gateway of the Hicom 300 that implements the interface between analogue data (audio, video) and Internet protocols is called the **Telephony Internet Server (TIS)**. In addition, TIS is able to integrate different kinds of videoconferencing (H.323 over LAN and H.320 over ISDN) and (ISDN-)telephony equipment (see figure 1). The several TIS components (call processing, gateway, dispatcher, feature processing etc.) are implemented as Windows NT services and can be administered and controlled via the Simple Network Management Protocol (SNMPv2) [17]. The management agent on TIS is implemented as a separate Windows NT service and uses the Windows NT SNMP extension agent facility. This feature allows the registration of a new management module (implemented as dynamic link library) without the need to recompile the agent. However, every time a new extension is added to the agent, the whole agent has to be restarted in order to access the new functionality. The TIS-specific part of the SNMP agent has a size of 250 kbytes and consists of about 150 variables and 20 tables; the TIS management information base (MIB) defines 6 types of asynchronous event messages.

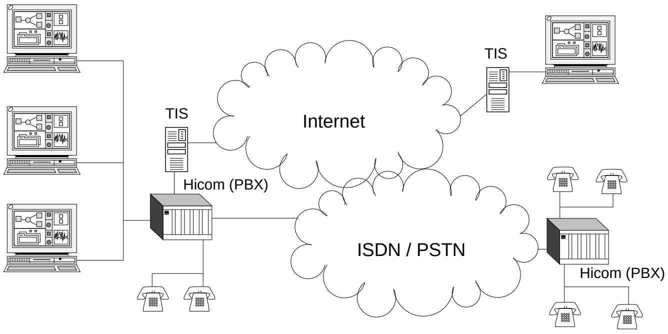

Figure 1: Siemens Telephony Internet Server: Architecture

On the side of the managing system, several Java-Applets exist for the Web-based management of TIS: They are downloaded from a WWW server and connect to the TIS management agent. The latter accesses the C++ based management instrumentation of the TIS and executes the monitoring and control of important variables such as the available and the peak bandwidth for a specific service type, the maximum and current number of calls and IP-specific settings (domain name, name servers, transmitted packets etc.). Management information relevant for accounting and billing purposes (e.g., caller and callee identification, length of call, used bandwidth etc.) can also be retrieved from TIS. Furthermore, it is possible to assign maximum bandwidths to specific connections and to modify them at runtime. Facilities for deleting and setting up calls and initiating/suspending/resuming the transfer of video streams are also available.

It is obvious that – particularly in large corporate networks –

access policies to these MIB variables have to be defined: Some variables should be readable/settable for local administrators while others might only be accessible to administrators responsible for the enterprise-wide Internet telephony server network. Although the SNMP *View-based Access Control Model (VACM)* [25] meets these requirements, it was not yet implemented in the current version of the TIS management agent. As the TIS management information base (MIB) has been implemented in a single module, there is no way of assigning specific access rules to MIB variables.

2.2 Management Requirements for Internet Telephony Servers

Traditionally, the administration and management of telephony systems and networks has been done by public carriers with specifically tailored – and often proprietary – frameworks, protocols and tools. Nowadays, the paradigm of voice/data integration implies that telecommunication systems are increasingly being managed with technologies stemming from the data communications world. This means that management of Internet telephony servers should rely on open, standardized middleware infrastructures in order to be compatible with already established data network management environments.

However, the very high requirements on any kind of telephony equipment with respect to scalability, availability, reliability and fault tolerance are rarely met by current data network management systems. This can be considered as one reason why telecom carriers are – despite the recent progress e.g., in SNMP-based management – still reluctant of deploying data communication management technologies to their domain. As Internet telephony servers are a special kind of telecommunication equipment, these requirements also apply to their management.

Thus, development frameworks serving as a basis for Internet telephony server management in large-scale corporate networks should provide:

- fine-grained security mechanisms (comparable to the SNMP VACM),
- a high degree of flexibility and adaptability,
- the ability to enhance or modify the amount of functionality at run-time,
- means to permit agents a large degree of autonomy without subordinating the target of global optimization to local optimization concerns,
- a way of enabling the cooperative solution of problems (e.g., problem determination tasks),
- access to standardized naming and directory services,
- simple update mechanisms for the agent software,
- persistent storage of agents, including their states, to guarantee a safe and consistent restart after a failure,
- an easy integration in already existing management systems,
- the support for Java-based management, thus yielding a high-degree of platform independence and increasing portability.

The above list of requirements is obviously not specific for Internet telephony servers but applies to any large-scale distributed environment with high availability and security restrictions. These servers can be regarded as a characteristic example of a highly distributed environment and have management requirements similar to other complex systems. We believe that distributed environments should be managed in a distributed way and focus in the next sections on a JavaBeans-based management approach.

2.3 Related Work

Distributed network, systems and application management is an active field of research motivated – among other – by the experiences with deploying centralized management systems (so-called *management platforms*) in large-scale corporate networks.

Management by Delegation (MbD) [5, 14, 15] defines a concept for delegating functionality from managing systems to agents in order to enhance them at run-time. The decision as to when this delegation should happen and which kind of functionality should be transferred to the agents is taken by the managing system, thus preventing autonomous decisions by the agents. **Cooperative agents** [11] rely on MbD and exhibit a certain degree of autonomy; they are able to receive event notifications from peers and can be grouped together in order to jointly achieve a task. Adding the concept of mobility to these agents yields **mobile agents** [12, 13]; their roaming capabilities allow them to move across networks in order to achieve specific, pre-defined tasks. However, the applicability of mobile agents is bound by security concerns; [6] and [24] discuss these aspects. The OMG Mobile Agent Systems Interoperability Facility (MASIF) [10] specifies an infrastructure for mobile agents in a CORBA [3] environment. **Mobile management agents** are designed to achieve administrative tasks on systems and software; while [1] discusses the advantages of applying mobile agents to management, [4] presents a Java-based environment for configurable and downloadable lightweight management applications.

3 Java Dynamic Management Kit

The **Java Dynamic Management Kit (JDMK)**[1], developed by Sun Microsystems [19, 20], promises to overcome many of the mentioned problems and supports some of the new requirements. JDMK was developed primarily for network management purposes. In the next sections, we will introduce the architecture and services of JDMK.

3.1 Architecture

JDMK represents a framework with corresponding tools, based on the JavaBeans specification [18], for the development of management applications and management agents. The base components of the architecture are summarized in figure 2. The Core Management Framework, M–Beans, C–Beans and different kinds of adaptors and services are the essential parts of JDMK.

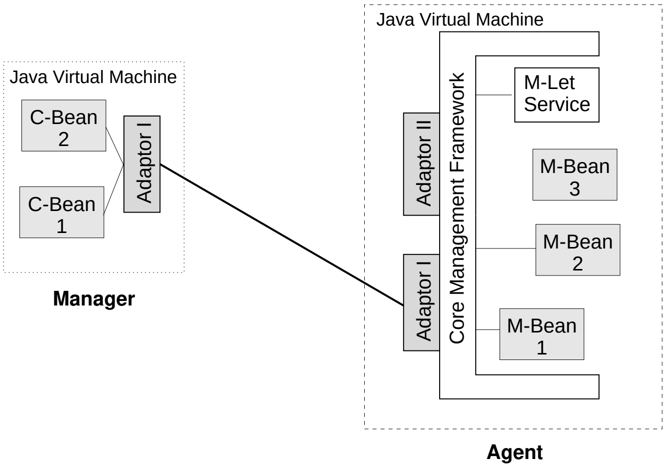

Figure 2: JDMK Architecture

M–Beans (Managed Beans) are Java objects implementing the “intelligence” and functionality of an agent. An M–Bean acts as a representative for one or more **managed objects (MOs)** or implements some functionality of a manager e.g., the preprocessing of data. With the aid of M–Beans it is possible to develop an agent that can be dynamically enhanced with new “management intelligence”. M–Beans are developed using a design pattern which is based on the JavaBeans component model. They are addressed with and uniquely identified by their object name. Names of M–Beans consist of the following parts: *classPart*[.*attribute*=*value*[,*attribute*=*value*]*]. Each of the parts can be set with user-defined values, especially the *classPart* does not need to be equivalent to the name of the implementation class of the M–Bean. The optional *attribute*=*value* pairs may be used to characterize M–Beans more precisely. All parts of the name can be used to define filtering rules for selecting special M–Beans (see also Filtering Service in Section 3.2).

[1] http://www.sun.com/software/java-dynamic

In order to use JDMK services or communication resources the M–Beans have to be registered at the so-called **Core Management Framework (CMF)** which represents a BeanBox for M–Beans. Only registered Beans can be accessed from outside of the CMF. The name of the M–Bean is used for the registration. The CMF generates the name if it was not set explicitly. The CMF together with its M–Beans represents the management agent. It is therefore the central interface for objects (M–Beans, services, . . .) which may be registered at the CMF from the agent itself or from a manager.

C–Beans (Client Beans) can be generated from M–Beans (or rather from the `.class`–files implementing the M–Beans) using a special compiler (`mogen`). They are proxy objects for remote M–Beans. M–Bean functions and data can be accessed by performing operations on C–Beans which are then propagated to the M–Bean. C–Beans use adaptors to communicate with their corresponding M–Bean. Together with their adaptors and additional management functionality they form the manager. An agent is also able to register C–Beans with its CMF. By doing this, the agent becomes a manager for that agent which implements the corresponding M–Beans. The strict separation between the manager and the agent role in protocol-based management architectures is therefore abolished in JDMK.

An **Adaptor** implements a special kind of a protocol, it is an interface for the CMF and for the agent. It is also realized as a Bean and therefore it is very easy to register adaptors at a CMF. With adaptors, manager and agent may be connected to each other or to other applications. At present RMI, HTTP, HTTP over SSL (HTTPS), IIOP, SNMP and a so-called HTML adaptor, which represents a Web–server, are available. This concept allows to communicate with the same JDMK agent by means of different protocols. It is not necessary to change the functionality or the code of the agent, the only thing to do is to register another adaptor. Should e.g., a Web–browser be used to connect to an agent the HTML adaptor must be registered at the CMF of the agent. This adaptor generates HTML pages for all M–Beans which are registered at the CMF. It is of course possible to use more than one adaptor at the same time.

3.2 Services and Development Tools

Besides of the base components of JDMK several services and tools exist to simplify the development of management applications and agents (cf. figure 3).

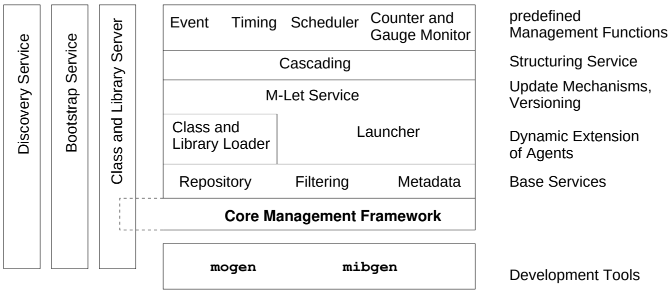

Figure 3: JDMK Services and Development Tools

For the registration process the **Repository Service** is used. Beans may be registered either as volatile or persistent within the CMF.

With the aid of the **Filtering Service** it is possible to define filtering rules selecting all M–Beans which are registered at a CMF that match the filtering rule. Such rules may be defined over methods, attributes with their concrete values and over object names and their single parts.

To determine the properties and methods which are supported by an M–Bean the **Metadata Service**, based on the Java Reflection API, can be used. The Metadata, the Repository and the Filtering Service are called **Base Services** of the CMF.

The **Discovery Service** is used to detect all active CMFs. Therefore an IP–broadcast is sent by the service and all CMFs that have registered a so-called *Discovery Responder* reply to the broadcast message.

Agent and manager are able to use different **Class Loaders** – even in parallel – to load the implementation classes from local or remote sites. To take advantage of native, (non–Java) dynamically linked libraries in M–Beans the **Library Loader Service** may be used. As a local or remote repository for class files and libraries the **Class and Library Server** is available. This server can either be used as a stand-alone application or – since it is realized as an M–Bean as well – registered with a CMF.

M–Let, Launcher and Bootstrap Service are used for the dynamic extension of agents, for update mechanisms and for bootstrapping.
The **M–Let Service (Management Applet Service)** offers the possibility to download and configure M–Beans dynamically. For this purpose a new HTML tag (`<MLET>`) is defined. The M–Let Service operates according to the following steps (see also figure 4):

1. The M–Let Service loads a HTML page from an URL from which it can obtain all the necessary information about the Beans to load (i.e., names, objects, code repository).
2. By using this information the M–Let Service is able to download the implementation classes of the M–Beans and to instantiate them.
3. Afterwards, the M–Let Service must register them at the CMF.

It is also possible to put version information inside the (`<MLET>`) tag and to use the M–Let Service for versioning.

The **Bootstrap Service**, which is realized as a stand-alone Java application, simplifies the distribution and instantiation of JDMK agents. The service is used to download implementation classes of an agent from a local or remote server. Therefore the Bootstrap Service initializes the CMF, starts the M–Let Service, loads the necessary classes, initializes, registers and starts all the required M–Beans and services of the agent.

With the **Cascading Service** it is possible to build hierarchical structures of master and sub agents. The master agent is the main interface towards the manager and hides the real location of sub agents.

Besides of these services JDMK also provides predefined management functions. The **Event Service** facilitates synchronous and asynchronous communications with unicast and multicast events. Triggering events, alarms or actions at predefined times is realized by the **Alarm** and **Scheduler Service**. **Counter Monitor** is used for the monitoring of discrete values and thresholds, **Gauge Monitor** for the monitoring of properties varying continuously.

Two compilers, `mogen` and `mibgen`, are delivered with JDMK as development tools. As mentioned, `mogen` is used to create C–Beans from M–Bean `.class` files.
If SNMP–MIB files are available for the managing device, the `mibgen` Compiler is able to use them to create M–Beans representing the MIB. The M–Beans have to be enlarged with functions e.g., implementing access to resources of the managed system. With these tools it is possible to build standalone or integrated SNMP agents. M–Beans of integrated agents are registered in the CMF and the integrated agent may therefore be reached through all protocols implemented in adaptors (e.g., RMI, IIOP, etc.) and also via SNMP. Communication with standalone agents is only possible over SNMP. It is not possible to distinguish from outside between a standalone JDMK agent and a classical SNMP agent.

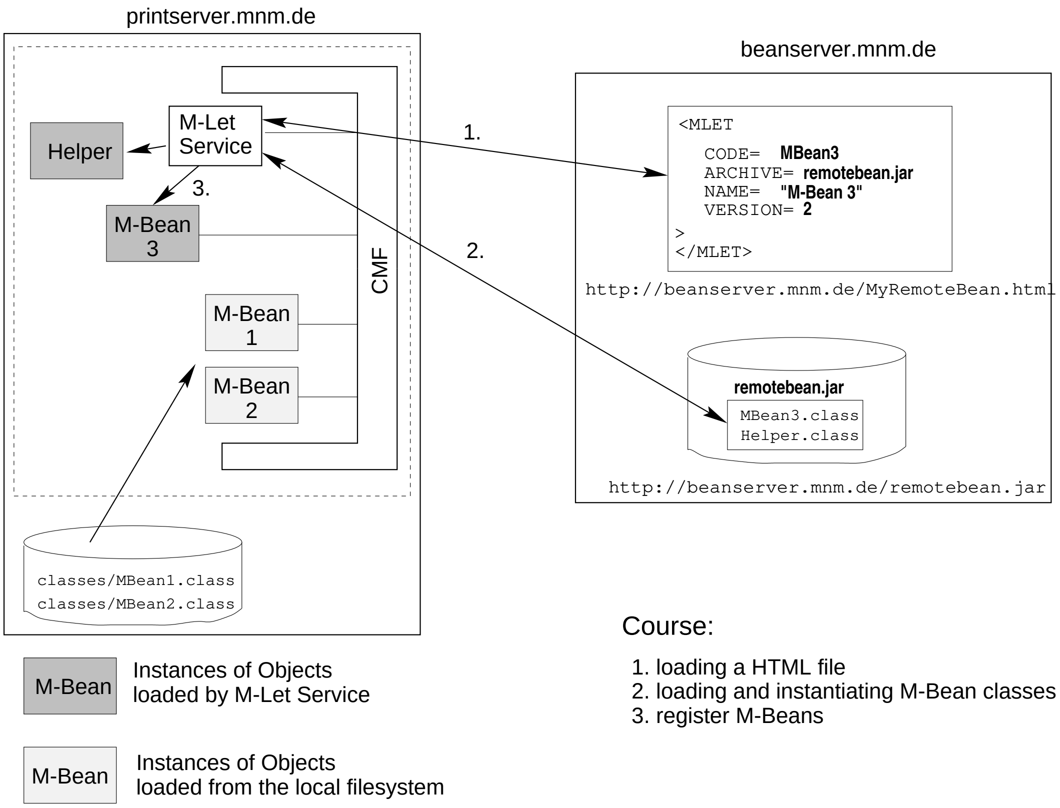

Figure 4: The JDMK Management Applet (M-Let) Service

4 TIS-Management with JDMK

JDMK has been used in this project because it offers interesting possibilities and meets many of the requirements for management application development frameworks.

4.1 Code Generation, Update Mechanisms and Native Code

To reduce time and costs of adaption and to minimize errors, as much code as possible should be generated automatically during the development of a TIS agent. Taking the TIS MIB as input, M–Beans are generated using the `mibgen` compiler. The conversion of the TIS MIB structure into corresponding M–Beans is performed automatically: The information mapping performed by `mibgen` is very similar to the JIDM specification translation algorithm for converting SNMP MIBs into OMG IDL descriptions [8]. Both JIDM and mibgen map MIBs to object-oriented languages. The whole MIB as a container for the groups, tables and variables is translated into an M–Bean. Each SNMP group becomes an M–Bean; every variable in a group is represented as an M–Bean property. SNMP table rows are translated into M–Beans; the variables contained in the rows are mapped to M–Bean properties, respectively. For detailed information on the conversion the reader is referred to [20].

A conversion example for a MIB structure is (partly) shown in figure 5. The left side of the figure depicts a very small part of the TIS MIB; the right side represents the appropriate M–Bean structure.

The group `GkStatistics` of the TIS MIB stores important statistical information of the Gatekeeper component of TIS (e.g., current number of calls, average length of calls in the last hour, etc.). The table `gkStatTable` stores Gatekeeper statistics of the last 48 hours. Figure 5 shows how these parts of the MIB are transformed in an M–Bean structure. The `mibgen` compiler generates M–Beans for the whole group `GkStatistics`, for the table (`TableGkStatTable`) and for each entry in this table. For each object that is readable and writable `get` and `set` functions are generated.

The generated M–Beans have to be enhanced with algorithms implementing the specific access functions for TIS. This is shown at the bottom of figure 5, e.g., `GkStatisticsImpl.java` extends `GkStatistics` and imple-

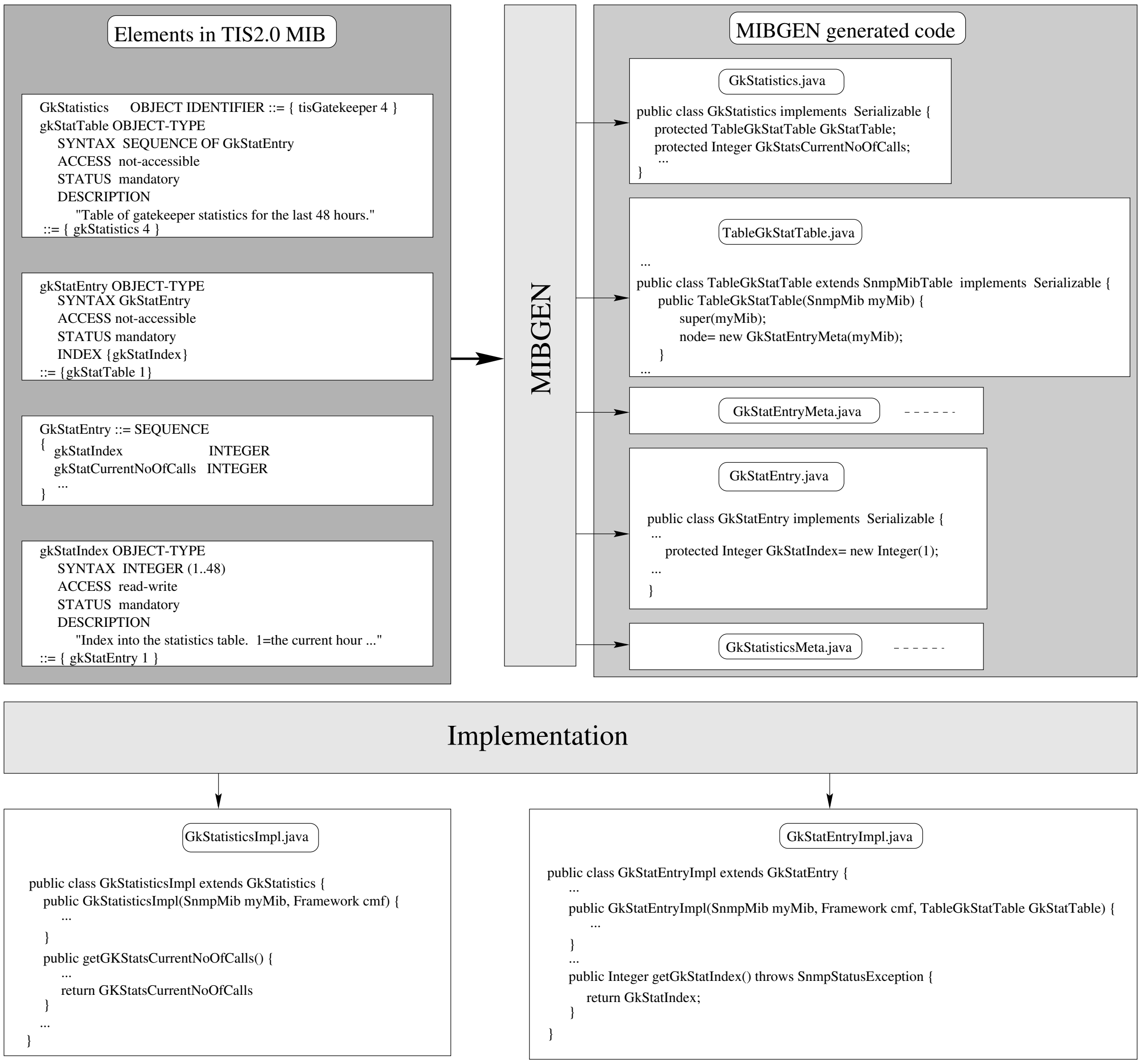

Figure 5: Generating M–Beans with `mibgen` (Example)

ments the `get` function for the current number of calls.

Nevertheless, an adaption after changes can be done relatively quickly as only the implementation of the changed M–Beans must be adapted. M–Beans of the agent can be replaced at run-time with the changed ones and it is possible to enhance the agent with additional M–Beans or objects.

If M–Beans have been changed or if a new version of the TIS agent has to be installed, the M–Let and the Launcher Service can be used for a dynamic update. The only thing that has to be done is to write the relevant classes at the `<MLET>` tag of the corresponding HTML file. The Launcher Service which uses the M–Let Service loads the new classes not yet installed or it replaces the whole TIS agent — if its version number has been increased.

Internally TIS is built upon various components (e.g., Gatekeeper, Administration Maintenance Server (AMS), . . .) which communicate by means of proprietary message format and inter-process communication (IPC). The required method calls for the IPC are combined into a dynamically linked library. In order to obtain management information from TIS components, the TIS agent must be able to construct and handle IPC messages. The dynamic link library was integrated for that purpose into the agent using the Java Native Interface (JNI) and is encapsulated by the Java class `Native Base` (cf. figure 6). The result is that the JDMK agent and its M–Beans are able to use Java method calls for the management of TIS and its components. Changing the MIB would not affect this interface.

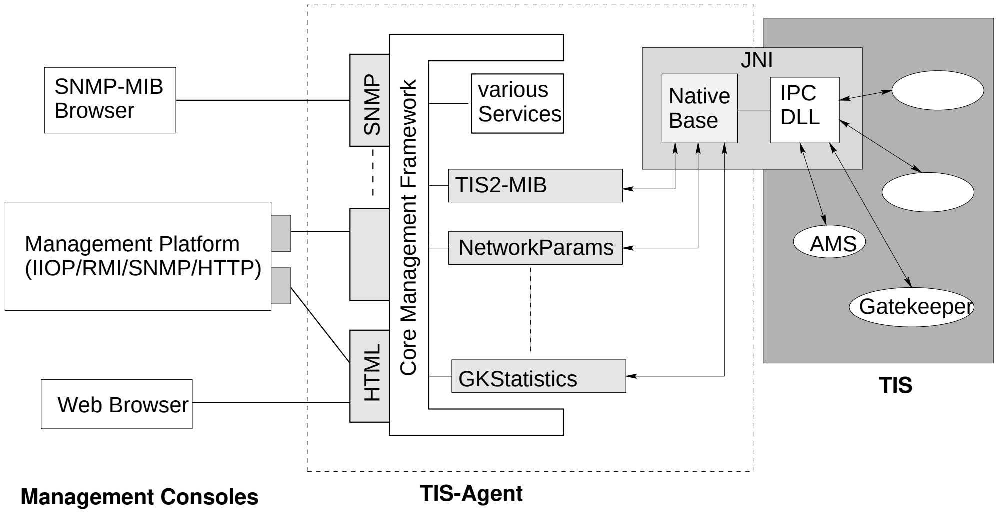

Figure 6: JDMK-based TIS Management: Architecture

4.2 Integration into existing Management Systems

A requirement on the TIS agent was that it should be possible to use a WWW browser as a management console; the agent should support the Web-based management paradigm. This demand can easily be fulfilled by registering the HTML adaptor at the CMF of the TIS agent. The HTML adaptor is accessed like a conventional WWW server, it dynamically creates HTML pages with all information and method interfaces available from the TIS agent.

Pages generated from the HTML adaptor do not always meet the requirements on modern user interfaces. Nevertheless, they provide a sound basis for further enhancements. In order to manage TIS from a WWW browser, it is sufficient to register the HTML adaptor with the TIS agent.

Besides the Web interface, it should be possible to access the TIS agent with SNMP or other protocols (e.g., RMI). This can also be easily achieved by registering the corresponding protocol adaptor.

A further requirement was that it should be possible to integrate the agent in a CORBA based management system. JDMK provides an IIOP adaptor; but this is not sufficient for the integration in a CORBA environment: The adaptor implements the IIOP protocol, but makes no use of any existing CORBAservices. In particular, it does not register the TIS agent or its M–Beans with the CORBA Naming Service or with the CORBA Interface Repository. The interface definition of the adaptor defines only simple `get` and `set` methods which are the only ones registered with the CORBA Interface Repository. It is thus necessary either to be aware of the whole internal structure of the TIS agent and all M–Beans or to considerably modify the implementation of the adaptor or the agent in order to manage TIS from a CORBA-based management system. It can be concluded that the IIOP adaptor serves only as IIOP-wrapper for the M–Bean access methods.

In our implementation of the TIS Agent the IIOP Adapter is registered without any changes. This shows that it is generally feasible to connect to a JDMK agent over IIOP. But a complete integration requires extensive changes and cannot be done automatically or semi–automatically.

The architecture for the management of TIS developed during the project is given in figure 6. `TIS2-MIB`, `Network-Params` and `GKStatistics` are examples of M–Beans which implement parts of the TIS MIB.

5 Evaluation of JDMK

This section evaluates JDMK with respect to its applicability to large enterprise management environments. We will thereby focus on three major problem domains which are of critical importance for a successful deployment and address the requirements identified in section 2.2.

5.1 Rapid Prototyping of flexible, dynamic Management Systems

As outlined in the previous sections, the primary strengths of JDMK lie in its capability of realizing flexible and highly distributed management systems.

Simple Web-based user interfaces can be generated automatically by using the HTML-Adaptor. JDMK-administered resources can easily be accessed from systems in other management architectures, because several adaptors for different management protocols are provided. It is thus possible to administer JDMK-based agents from SNMP management platforms through the SNMP adaptor. The toolkit also enables the development of adaptors for new, not yet supported protocols. The adaptor concept is helpful if agents should support multiple management protocols simultaneously e.g., information provided by a specific agent should be accessible not only from an SNMP-based management platform but also from a Web–browser via HTTP.

Agents developed with JDMK can be enhanced and modified at run-time, thus yielding the opportunity of delegating management tasks to them via the push model. Furthermore, these agents are also allowed to initiate the download of management functionality by themselves (pull model). These additional services are only transferred to the agent if needed. If the agent detects an error condition in the managed resource, it can request additional management functionality e.g., specific problem determination and error-correction services. Under regular conditions, tasks of the management system – such as the correlation of different variables – may already be carried out by the agent, thus preventing the exchange of large amounts of data between the managing system and the agent. The JavaBeans concept and the simple registration mechanism of Beans with the CMF of the agent facilitate quick response times when status changes are detected. The services provided by JDMK enable distribution and update mechanisms to enhance agents with additional functionality residing in centralized code repositories.

The access to resource-specific (and thus non-portable) management instrumentation is eased by the integration of appropriate dynamic link libraries through the Java Native Interface. In case that no Java Virtual Machine is available for a resource, a JDMK-based agent installed on another system may act as a proxy for the resource.

JDMK also provides mechanisms for the persistent storage of M-Beans, thus enabling the agent components to remain close to the resource and eliminating the need of downloading them from remote servers. However, only the basic serialization mechanisms of Java are available. Mechanisms for the recovery from system crashes or errors within the agent (like incremental backups of the agent context and software) require the extension of the agent and the CMF by the developer.

In summary, we believe that JDMK has a strong potential for the rapid development of highly distributed management environments and provides with its several protocol adaptors a good basis for today's heterogeneous management environments.

5.2 Suitability for large IT infrastructures

We will now analyze how well JDMK can be integrated with large-scale distributed environments such as CORBA and discuss whether it can make use of the large amount of already specified CORBAservices.

As it was pointed out in section 4.2, JDMK does neither provide standardized directory and naming services nor mechanisms for making use of CORBAservices. Features for achieving location transparency (like the CORBA Interoperable Object References) are also not available: An M–Bean is identified by a protocol identificator, the host address and port number, and its object name. These parameters and some knowledge about the registered beans in a given CMF must be present in order to make use of the M–Beans.

The scope of the metadata service that can be used to retrieve information on registered M–Beans is limited to *a single* CMF, i.e., there is no global metadata service such as the CORBA Interface Repository in conjunction with the Dynamic Invocation Interface or the Trader Object Service. Consequently, the only way of finding the currently active CMFs is the JDMK Discovery Service which sends a broadcast message that is answered by all running agents. In order to then find the agents that implement e.g., a specific method, every agent has to be queried individually via the JDMK Filtering Service. Therefore, JDMK services have a much smaller scope than CORBAservices.

The establishing of domains and the structuring of the agents in functional groups is not supported by the development environment and has to be done by the developer.

Inter-agent communication is only feasible between master and sub agents. As sub agents are not supposed to interact with each other, JDMK cannot serve as a communication infrastructure for cooperating agents. Another issue is that JDMK-based agents are not able to migrate in a network. Therefore, JDMK agents are not mobile in terms of the OMG Mobile Agent Systems Interoperability Facility (MASIF). They are installed on a host and are unable to "visit" another one. As JDMK agents are not designed according to the MASIF specification, it is not – or only with great difficulties – possible to integrate them in a CORBA/MASIF-based management system.

The above mentioned conceptual weaknesses make it hard to develop management applications for large IT infrastructures where a high number of different JDMK-based agents are needed. The JDMK services are useful for small, local environments where the amount and degree of diversity of the agents are restricted. Due to the absence of focus on large systems, the scalability of JDMK-based solutions may be critical at the current stage of the toolkit.

5.3 Security Aspects

JDMK does not have a homogeneous security concept for its different protocol adaptors; instead, developers need to be aware of the different security mechanisms to implement comprehensive security for agents that support different protocol adaptors.

The SNMP adaptor relies on a file containing access control lists to determine which management systems have the right to read or modify specific parts of the MIB. Although this can be considered as an enhancement compared to the (password-based) mechanism of the early SNMP, modern fine-grained SNMP security mechanisms like VACM [25] are not yet supported. As the authentication of remote systems is based on their IP address, the agent is vulnerable with respect to IP-spoofing attacks.

The authentication method of the HTTP/HTML adaptors are login/password combinations. As sensitive data is exchanged unencrypted, it is not possible to implement secure HTTP-based management solutions.

The RMI and IIOP adaptors do not support authentication and access control.

The only way of enabling secure authentication is based on the HTTPS (HTTP over SSL) adaptor which allows the exchange of cryptographically secure certificates. The appropriate access control system must then be implemented by the developer.

We believe that the current security mechanisms of JDMK are insufficient because developers still have to implement a large part of the security mechanisms themselves. Furthermore, the large differences between the various security mechanisms are not yet shielded behind a comprehensive security architecture. Agents that support multiple management protocols simultaneously thus have to implement several security models.

6 Java Management Extensions

In mid June 1999 the draft version of the new **Java Management Extensions (JMX)** specification [22] has been released for public review. JMX integrates the former developments within the scope of the Java Management API (JMAPI) and JDMK into a Java-based management framework. The specification does not only focus on the agent part of the management system (as it was the case with JDMK) but will also specify the manager part. However, in the current version of the specification the JMX manager is left blank. JDMK can be considered as an integral part of JMX. We will therefore provide a short overview over the most recent developments and address the question whether the critical issues of JDMK (as described in the previous section) also apply to JMX. However, please note that the JMX specification is not yet in a final state. Information on the latest developments can be found at the Sun website.

The JMX architecture it is very similar to JDMK depicted in figure 2. JMX distinguishes between **manager**, **agent** and **instrumentation** levels: Within an agent, M–Beans responsible for making a resource manageable (including the generation of event notifications) reside at the instrumentation level. A manageable resource in JMX can be an application, a service implementation or a device. The instrumentation is made accessible to JMX managers (forming the manager level) through the agent level which provides a communications interface, a set of standard services and a run-time environment. The Core Management Framework of JDMK has been renamed to **MBean server**.

One of the major changes in JMX refers to MBeans[2] that have been categorized into four distinct classes: The **standard MBean** is a subset of the M–Beans known from JDMK; its interface consists of the method names. In contrast, **Dynamic MBeans** expose their properties and operations at run-time via defined operations that return all the attribute and operation signatures. The purpose of dynamic MBeans is to ease the instrumentation of existing (legacy) managed resources. An **open MBean** is a dynamic MBean which offers of universal, predefined data types and functions and provides – in addition – detailed meta-information regarding its interface (`MBeanInfo`). The goal is to have "self-describing" MBeans that can easily be used when needed. The **model MBean** – another kind of dynamic MBean – is a generic, configurable management template for managed resources. It can be instantiated either by the JMX Agent or by other MBeans or even by the managed resource itself. At creation time, the method signatures and the set of attributes exposed by the model MBean can be defined in XML, OMG IDL or Java. The model MBean also offers the option of persistent storage.

The adaptor concept of JDMK is now divided up into **protocol adaptors** and **connectors**. Protocol adaptors are used to link an JMX agent with non-JMX compliant management applications (i.e., SNMP, Common Information Model (CIM) [2] or proprietary). The connector – in contrast – is used by a remote JMX-enabled management application (i.e., developed using JMX manager services) to connect to a JMX agent.

JMX extends the event mechanism of JDMK into a **notifi-**

[2] The dash in the word "M–Bean" has been removed.

cation model which enables a listener to register only once and still receives all events. To select special kinds of notifications a **notification filter** is offered. A management application can be notified whenever a value of a given MBean attribute changes using the **attribute change notification** or whenever a MBean is created or deleted.

The services described in section 3.2 are also available in JMX. The predefined management services `timer`, `counter` and `gauge monitor` have been integrated into the new notification model. In addition, a new **String monitor** has been defined to control changes of string objects.

For the integration with existing management solutions JMX offers additional **management protocol APIs** and includes an open interface that any vendor can use. Currently, an **SNMP API** [23] and **CIM/WBEM APIs** [21] are defined and implemented. The CIM client API deals with the transfer of data between JMX-based management applications and CIM Object Managers (i.e., CIM-compliant management systems); the provider API describes the interface between a CIM Object Manager and JMX agents.

The final JMX specification will provide both a reference implementation and a compatibility test suite. The former is intended to allow developers to prototype management applications easily whereas the latter checks if a implementation conforms to the specification. However, none of them are available yet.

Based on the current draft of the JMX specification, we have to state that the shortcomings of JDMK that have been identified in section 5 also apply to JMX. To which degree the review process of the specification will eliminate the weaknesses of the current JMX version is still an open question. This particularly applies to the manager side of JMX, which is yet unspecified.

7 Conclusion and Outlook

This paper has described a case study for the dynamic management of Internet telephony servers based on JavaBeans and JDMK. We have discussed our implementation concept with extending the management agent of the Siemens Telephony Internet Server. Our work was motivated by the increasing demand for scalable and reliable solutions which allow the extension of management agents at runtime. The experiences gained in this project allow the evaluation of the applicability of JDMK for managing large-scale enterprise networks and can be summarized as follows:

The development environment permits rapid prototyping and is easy to use; the transfer of lightweight applications (implemented as JavaBeans) to management agents at runtime works very well: JDMK supports both push and pull models and enables agents to acquire additional functionality, thus improving their (albeit limited) autonomy. However, JDMK-based agents are neither able to cooperate nor can they migrate across networks. Therefore, they cannot be considered as mobile agents; according to the definitions given in section 2.3, JDMK is best described as a development framework for Java-based *Management by Delegation*. At its current stage (version 3 beta 2), JDMK is a powerful toolkit for the development of management agents that can be accessed and modified through several different communication mechanisms (CORBA/IIOP, RMI, HTTP, SNMP, HTTPS). However, the differences between these communication infrastructures are not hidden by the development environment and thus have to be addressed by the developer.

The usability of management systems – especially in an enterprise-wide context – depends to a high degree on the security features of the underlying middleware. However, the JDMK security mechanisms are yet unsatisfactory because the different mechanisms of the underlying communication protocols/infrastructures have not yet been integrated into a common security architecture. It therefore depends on the type of the underlying protocol whether e.g., encryption is available and how access control is handled. Another critical issue is the absence of services to obtain meta-information on the deployed agents (like the CORBA interface repository and the naming and trader services): The services to obtain information regarding the whole set of agents in a JDMK environment lack scalability because they can only be applied to *a single* Core Management Framework, thus preventing a “global” view on the agents.

JDMK is not positioned as a stand-alone management framework but as an integral component of the *Java Management Extensions (JMX)*, the emerging Java-based management framework that is currently developed by Sun Microsystems, Inc. and leading companies in the systems management field (such as IBM, Computer Associates, BullSoft, Tibco, Xylan and Powerware) through the Java community process. Even if the JMX specification is yet incomplete, it can be expected that the future development of JMX will eliminate the weaknesses that have been identified during our work.

Acknowledgment

The authors wish 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. Its webserver is located at `http://wwwmnmteam.informatik.uni-muenchen.de`.

References

[1] Andrzej Bieszczad, Bernard Pagurek, and Tony White. Mobile Agents for Network Management. *IEEE Communication Surveys*, 1(1), 1998. `http://www.comsoc.org/pubs/surveys/4q98issue/bies.html`.

[2] Common Information Model (CIM) Version 2.2. Specification, Distributed Management Task Force, June 1999.

[3] The Common Object Request Broker: Architecture and Specification. OMG Specification Revision 2.2, Object Management Group, February 1998.

[4] M. Feridun, W. Kasteleijn, and J. Krause. Distributed Management with Mobile Components. IBM Research Report RZ 3102, IBM Research Division, Zurich Research Laboratory, February 1999.

[5] G. Goldszmit and Y. Yemini. Distributed Managment by Delegation. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, June 1995.

[6] Michael S. Greenberg and Jennifer C. Byington. Mobile Agents and Security. *IEEE Communications Magazine*, 36(7):76–85, July 1998.

[7] Nicholas R. Jennings and Michael J. Wooldridge. *Agent Technology – Foundations, Applications and Markets*. Springer, Berlin, Heidelberg, New York, 1998.

[8] Inter-Domain Management: Specification Translation. Open Group Preliminary Specification P509, Open Group, March 1997.

[9] H. Knöchlein. Management eines Internet Telefonie Servers mittels JDMK . Diploma thesis, Technische Universität München, February 1999.

[10] Mobile Agent System Interoperability Facilities Specification. OMG TC Document orbos/98-03-09, Object Management Group, March 1998.

[11] M.-A. Mountzia. *Flexible Agents in Integrated Network and Systems Management*. PhD thesis, December 1997.

[12] Vu Anh Pham and Ahmed Karmouch. Mobile Software Agents: An Overview. *IEEE Communications Magazine*, 36(7):26–37, July 1998.

[13] K. Rothermel and F. Hohl, editors. *Mobile Agents (MA '98)*, volume 1477 of *LNCS*, Berlin; Heidelberg, 1998. Springer.

[14] Jürgen Schönwälder. *Netzwerkmanagement mit programmierbaren, kooperierenden Agenten*. PhD thesis, Technische Universität Braunschweig, March 1996.

[15] Jürgen Schönwälder. Network management by delegation - from research prototypes towards standards. In *8th Joint European Networking Conference (JENC8)*, Edinburgh, May 1997.

[16] Siemens AG. The convergence of voice and data. White Paper, 1998. `http://www.siemens.se/siemensab/communications/itnet/papers.html`.

[17] SNMPv2 Working Group, J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. Protocol operations for version 2 of the Simple Network Management Protocol (SNMPv2). RFC 1905, January 1996.

[18] Sun Microsystems, Inc. JavaBeans, Version 1.01. Technical Specification, Sun Microsystems, Inc., Palo Alto, CA, July 1997. `http://www.javasoft.com/beans/docs/spec.html`.

[19] Sun Microsystems, Inc. Java Dynamic Management Kit. White Paper, Sun Microsystems, Inc., Palo Alto, CA, 1998. `http://www.sun.com/software/java-dynamic/wp-jdmk/`.

[20] Sun Microsystems, Inc. Java Dynamic Management Kit 3.0 (beta). Programming Guide, Sun Microsystems, Inc., Palo Alto, CA, August 1998.

[21] Sun Microsystems, Inc. Java Mangement Extensions CIM/WBEM APIs. Preliminary Specification Draft 1.9, Sun Microsystems, Inc., Palo Alto, CA, June 1999.

[22] Sun Microsystems, Inc. Java Mangement Extensions (JMX). Preliminary Specification Draft 1.9, Sun Microsystems, Inc., Palo Alto, CA, June 1999.

[23] Sun Microsystems, Inc. Java Mangement Extensions SNMP Manager API. Preliminary Specification Draft 1.9, Sun Microsystems, Inc., Palo Alto, CA, June 1999.

[24] Giovanni Vigna, editor. *Mobile Agents and Security*, number 1419 in LNCS, Berlin, Heidelberg, 1998. Springer.

[25] B. Wijnen, R. Presuhn, and K. McCloghrie. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). RFC 2275, January 1998.