Proceedings of *Trends in Distributed Systems: Towards a Universal Service Market (USM 2000)*, Munich, Germany, September 2000

# Monitoring Quality of Service across Organizational Boundaries

Rainer Hauck and Helmut Reiser

Munich Network Management Team,
University of Munich, Dept. of CS, Oettingenstr. 67, D-80538 Munich, Germany
{hauck|reiser}@informatik.uni-muenchen.de

**Abstract** *In a customer/provider relationship a provider offers services to a customer who pays for using them. To control service delivery certain Quality of Service (QoS) parameters have to be agreed through so called Service Level Agreements (SLAs). Monitoring of some of these QoS parameters only makes sense when done from a customer perspective. Therefore, the data collection has to take place at the customer site, which means at least one organizational boundary has to be crossed. In a cooperation project we developed a flexible and extensible agent for that purpose using the Java Dynamic Management Kit (JDMK) and the Application Response Measurement API (ARM). Based on our experiences with the prototype implementation this paper analyses the suitability of JDMK and ARM for the monitoring of QoS parameters and SLAs and for building scalable and flexible management systems in large–scale enterprise networks.*

**Keywords:**
Quality of Service, Intra-/Extranet, Application Response Measurement, Service Level Agreement, Application Management, JDMK, JMX

## 1 Introduction

In a typical customer/provider scenario a customer demands - and pays for - a certain Quality of Service (QoS) laid down in so-called Service Level Agreements (SLAs). In recent years growing demand from customers to monitor the performance of their network services can be realized. Poor performance is no longer tolerated. Instead customers now can receive discounts or even have the right to cancel the contract and choose a different provider. But network parameters like numbers of IP packets dropped or the available bandwidth at a certain point in time are not very meaningful to the vast majority of customers. So it seems preferable not to monitor network performance but service performance instead. In order to observe the quality of delivered services it is necessary to negotiate Quality of Service (QoS) parameters as well as means for the measurement and evaluation of these parameters. The monitoring of the QoS parameters has to be done by an agent running at the customer site, because this is the only way to monitor actual service usage of the customer. The fact that an agent belonging to a provider must be running at the site of a customer implies strong security requirements. Due to numerous changes to SLAs in course of time, the agent has to be flexible and extensible in order to allow easy updating.

The paper shows how concepts and services of the Java Dynamic Management Kit (JDMK) can be used for building such an architecture. It also points out weaknesses and deficiencies JDMK still has to deal with. As an example of how monitoring of QoS parameters can be done, a web browser was instrumented using the Application Response Measurement (ARM) API to deliver information about the actual response times a customer is experiencing.

The paper is structured as follows: Section 2 gives an overview about the scenario the paper deals with and shows the requirements a management solution for this scenario has to fulfill. In section 3 JDMK and the ARM API which are both used in building our solution are introduced. Section 4 describes how we applied JDMK and ARM to the problem domain. An example how actual QoS parameters can be obtained is shown. Section 5 summarizes our experiences and evaluates how JDMK and ARM are suitable for monitoring QoS parameters in large enterprise environments. The *Java Management Extensions (JMX)* specification 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 what degree JMX alleviates the shortcomings of JDMK. Section 7 concludes the paper.

## 2 Scenario: Outsourcing of Extranets

An exemplary scenario – which this paper is based on – is a car manufacturer (customer) which enables all its dealers (more than 1000) to use special applications within its corporate network (CN) (e.g., Online Ordering of cars). The dealers build a so called extranet (EN). Besides the specific applications the EN enables connections to the worldwide Internet. The provider implements the EN infrastructure with all the demanded services on the customers behalf. It is also responsible for the management of the extranet with all services it offers. It has to be remarked that the provider is not responsible for applications offered by the customer.

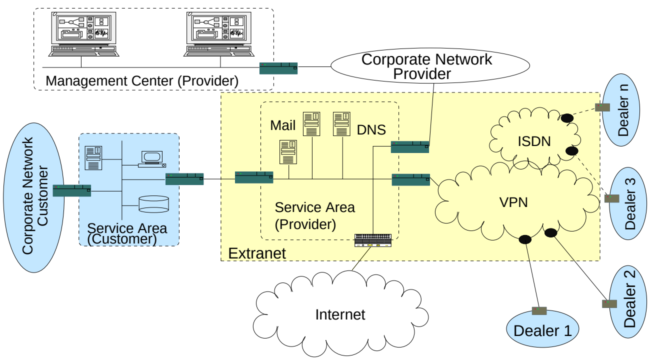


**Figure 1.** Extranet Solution for a Car Manufacturer

The corresponding infrastructure is presented in figure 1. Dealers are connected with a Point of Presence (PoP) of the provider over leased lines or ISDN dial-up lines. They form a virtual private network (VPN) on top of the infrastructure of the provider, which is also used by other customers. In the Service Area (SA) of the provider reside servers for the services rendered by the provider (e.g. Mail, DNS, Authentication Service, ...)

and a well defined access point to the Internet. If a dealer wants to use a service of the manufacturer it will be routed into the SA of the car manufacturer. In this configuration dealers use services both in the SA of the provider and in the SA of the customer.

### 2.1 Management Requirements for Monitoring Quality of Service

In order to observe the quality of delivered services it is necessary to negotiate QoS parameters between customer and provider. As mentioned before, it is important that these parameters are meaningful to the customer and reflect its expectations. They are part of Service Level Agreements which essentially are a contractual relationship between a customer and a provider. SLAs are used – among other things – for billing purposes. If a service is not delivered with the quality that was agreed in the SLA the customer may get a discount. Therefore the SLA (and therefore the QoS parameters) have to be supervised by the management system of the provider. Further the provider is obliged by the SLA to report compliance with agreed QoS parameters. Examples for such QoS parameters are the response time, the connectivity or the availability not of a network connection but of a certain service from a customer's point of view. Thus these parameters have to be measured and valued in the customer environment. Most of the QoS parameters (e.g. the connectivity or the availability) of a service are only defined if the customer actually tries to use this certain service. For the monitoring of such parameters it is necessary for the management system of the provider to be able to measure from the side of the customer. In our scenario that means an agent which performs the monitoring of QoS parameters must be installed at a dealer's host. Thus the agent has to cross the organizational boundary between the customer and the provider. The customer will allow the provider to do so only if high security requirements are met. Additionally a customer will only accept such a solution if no additional or only minimal costs are caused. This especially causes problems for customers which are connected via dial-up lines, because the initiation of a connection solely for management purposes cannot be tolerated in this case. Furthermore it means the agent has to do its measures locally as far as possible and transfered results should be as small as possible. As requirements for services, QoS parameters and SLAs could change in course of time the architecture has to be flexible and must be able to react on such changes quickly. In large–scale corporate networks as described in the scenario, well–scaling management systems are absolutely necessary. In addition, a lot of different hardware and software at the customer side, requires a high degree of platform independence for the corresponding agent.

### 2.2 Related Work

Distributed network, systems and application management is an active field of research motivated – among others – by the experiences with deploying centralized management systems in large-scale corporate networks. To overcome deficiencies of centralized management systems a lot of research has been done.

**Management by Delegation (MbD)** [5,15,14] defines a concept for delegating functionality from managing systems to agents in order to enhance them at runtime. The decision 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. **Flexible 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. In recent years **mobile agents** [12,13] which add the concept of mobility to flexible agents or MbD have been investigated. Their roaming capabilities allow them to move across networks in order to achieve specific, predefined tasks. However, the applicability of mobile agents is bound by security concerns; [6] and [23] discuss these aspects. **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.

The following approaches for the monitoring of application response times are commonly used today: First **network monitors** like the *Tivoli Netview Performance Monitor*[1] can be used to monitor traffic on the network. However the packets must be correlated to actual user transactions in order to calculate response times, which is quite a complex task [16]. Moreover, it is very unlikely that a customer will allow a provider to place a network monitor in his corporate network. A different approach is by using **active probes** like done by *Geyer & Weinig's GW-TEL INFRA-XS*[2]. Here an agent at the remote site actively initiates transactions and measures the response time. The problem with this solution is that it does not monitor the actual transactions of the user but some test transactions and that it increases network and server load.

## 3 Framework and Development Tools

The Java Dynamic Management Kit (JDMK)[3], developed by Sun Microsystems [18,19], promises to overcome many of the problems mentioned and supports some new requirements. Application Response Measurement API (ARM) is a valuable standard for the application perfomance instrumentation. In the next sections, we will introduce the architecture and services of JDMK and the concepts and use of ARM.

### 3.1 Architecture and Services of JDMK

JDMK represents a framework with corresponding tools, based on the JavaBeans specification [17], for the development of management applications and management agents. The base components of the architecture are shown in figure 2.

M–Beans (Managed Beans) are Java objects implementing the intelligence and the functionality of an agent.

[1] http://www.tivoli.com/products/index/netview_perfmon/
[2] http://www.gwtel.de/
[3] http://www.sun.com/software/java-dynamic/

They act as a representative for one or more managed objects (MOs). In order to use JDMK services or communication resources M–Beans have to be registered at the so called Core Management Framework (CMF) . Only registered Beans can be accessed from outside the CMF. The CMF together with its M–Beans represents the management agent. C–Beans (Client Beans) can be generated from M–Beans using a special compiler. C–Beans are proxy objects for remote M–Beans. 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 hence for the agent. At present RMI, HTTP, HTTPS (HTTP over SSL), 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 a different adaptor. Of course it is possible to use more than one adaptor at the same time.

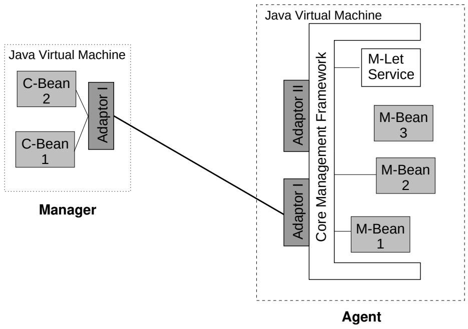


**Figure 2.** JDMK Architecture

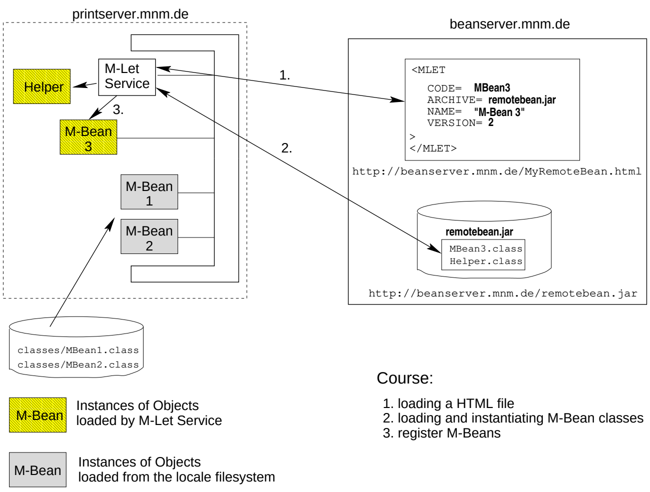


**Figure 3.** JDMK Management Applet (M-Let) Service

Besides the base components of JDMK several services and tools exist to simplify the development of management applications and agents. Services relevant for the project are briefly explained in the following paragraphs.

Beans may be registered with the aid of Repository Service either as volatile or persistent within the CMF. The Discovery Service is used to detect all active CMFs. To determine the properties and methods which are supported by an M–Bean the Metadata Service, can be used. The Class and Library Server serves as a local or remote repository for class files and libraries. M–Let, Launcher and Bootstrap Service are used for 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 (cf. figure 3). For this purpose a new HTML tag (`<MLET>`) is defined. First M–Let Service loads a HTML page from an URL from which it can obtain all the necessary information about the Beans to load. Then M–Let Service is able to download the implementation classes of the M–Beans and to instantiate them. Afterwards, the M–Let Service must register them with the CMF. It is also possible to put version information inside an (`<MLET>`) tag and thus use the M–Let Service for versioning. The Bootstrap Service simplifies the distribution and instantiation of JDMK agents. This service is used to download implementation classes. Therefore the Bootstrap Service initializes the CMF, starts the M–Let Service, loads the necessary classes, initializes, registers and starts all required M–Beans and services of the agent.

Besides `mogen` compiler which is used to create C–Beans there is `mibgen` compiler for developing a JDMK based SNMP agent for a device. If SNMP–MIB files are available for a managed device, `mibgen` 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.

### 3.2 The Application Response Measurement API (ARM)

The Application Response Measurement API (ARM) [2] has been developed in 1996 in a joint initiative of Tivoli Systems and Hewlett-Packard. Later that year HP, Tivoli and 13 more companies established the ARM Working Group of the Computer Measurement Group (CMG)[4] to continue development and promotion of the API. It promises to allow transaction based monitoring of response times in a distributed and heterogeneous environment. Work on version 2.0 of the API was finished in November 1997. In January 1999 the ARM API version 2.0 was adopted by the Open Group as its technical standard for application performance instrumentation.

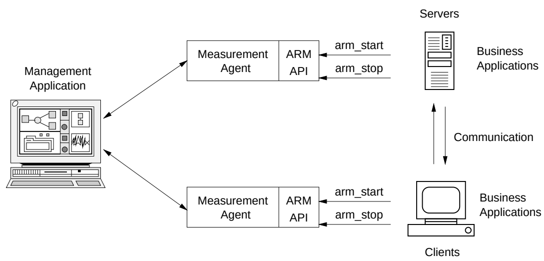


**Figure 4.** Using the ARM API

To achieve the goals mentioned above a simple API was defined that is supposed to be implemented by management tools. The applications to be monitored have to be instrumented to call the API whenever a transaction starts or stops. Actual performance monitoring is done by using management tools that provide measurement agents implementing the API. These are linked against the application to be monitored and thus are called whenever a transaction is initiated or ended. (see figure 4). An important feature added with version 2.0 of the API is correlation of transactions. Often

[4] http://www.cmg.org/regions/cmgarmw/index.html

one transaction visible to the user consists of a number of subtransactions. When indicating the start of a new transaction, a data structure called a correlator can be requested from the measurement agent. When indicating the start of a subtransaction, this correlator can be used to inform the measurement agent about the existence and identity of a parent transaction.

## 4 Monitoring of QoS using JDMK and ARM

In an industrial cooperation project with DeTeSystem [8] a prototypical management solution was developed that fulfills the requirements mentioned in chapter 2.1. In this project JDMK (version 3.0 beta 2) and ARM (version 2.0) were used for the monitoring of QoS parameters.

### 4.1 Architecture

As seen before there is a strong requirement for Service Level Monitoring to be considered from a customer's perspective. QoS parameters of SLAs are only reasonable if the user actually tries to use a service. If it does not try to use a special service there cannot be a violation of an SLA caused by the provider. Through the use of Flexible Management Agents (FMA) positioned in the corporate network of the customer, our solution enables the measurement of the actual Service Levels provided to the customer.

In order to avoid additional costs for a customer connected by a dial-up link the following solution was examined: The agent transmits the collected information to the manager only when the link is up and free. In addition packets from the agent do not affect the idle timer responsible for the disconnection of the link if it was unused for a predefined period of time. That would mean that only the (otherwise useless) timeout intervals are used for management traffic. However, the needed functionality is not available in ISDN routers commonly used today, as there is no way to distinguish between packets that should affect the idle timer and packets that should not. As a trade-off the following solution was chosen: Management traffic is created only when the link is up. Although being suboptimal this approach avoids establishing a connection solely for management purposes and thus reduces the additional cost to a minimum. To check whether the link is up or down, some kind of status information needs to be available. Two different solutions have been examined. Many routers send SNMP traps whenever a link is established or goes down. The router can be configured to deliver these traps to the agent. So the agent gets informed immediately about any change in link status. As not all routers send traps concerning the link status the second approach might be necessary: If the agent has management information to transmit, it polls the variable `ifOperStatus` (operational state of the interface) from MIB-II [10] on a regularly basis. Both variants mean that from the perspective of the router, the agent acts in the role of a SNMP manager. As a relatively high polling interval can be chosen (about half of the idle timeout value for the link) the additional load induced by polling is minimal.

Every time the SLAs change or new means of measurement are needed, new functionality must be installed in the corporate network of the customer. As the customer network is typically located far from the management center of the provider a solution is necessary that allows dynamic download of new functionality. This requirement is one of the main reasons why the presented solution is based on JDMK. The CMF has to be installed only once at the customer site and from then on all the functionality needed can easily be downloaded as an M-Bean using e.g., the M-Let Service. To prevent data from being accidentally destroyed, the agent can regularly be serialized to local disk. When the system comes up – for example recovering from a system crash – the JDMK Bootstrap Service tries to load the local copy of the agent before it contacts the central repository. Of course this does not prevent a customer from explicitly deleting the information collected by the agent before it is transmitted to the manager. Another benefit of using the JDMK was its ability to use different adaptors. The provider uses a management platform and also wants to use web browsers to configure the agents. By using JDMK's SNMP and HTML adaptors this could be achieved easily. A manager application was built to receive the performance data sent by the agents using the RMI adaptor. In the current version it writes the data to a database where management tools can access it.

### 4.2 Quality of Service Measurement

There are many different ways Service Level Agreements might be defined. Typical representatives are the availability of IP connectivity and the time needed by a server to fulfill a user request. In the given scenario there is the need to measure the response times of web servers in the service areas of both the provider and the customer. The transactions of interest are the download of web pages by the dealers. In order to learn about the actual response times the user experiences, a web client (Lynx) was instrumented with calls to the ARM API. Figure 5 shows the transactions to be monitored.

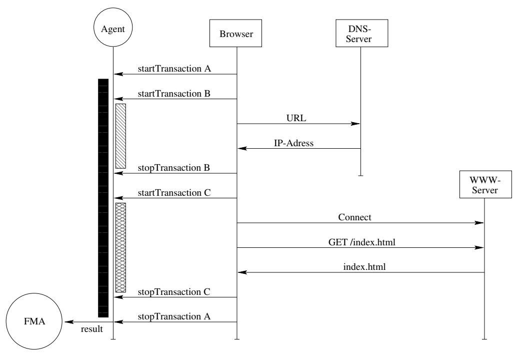


**Figure 5.** ARM instrumentation of a web browser

Most important is transaction A which spans the whole process of loading the page. To allow the manager the localization of failures or bottlenecks two subtransactions are defined. Subtransaction B covers the part of contacting the DNS for hostname resolution and subtransaction C covers the actual download of the page. The measurement agent forwards the results to the FMA which can do some preprocessing on the data and transmits it to the manager whenever the link is up. If the FMA receives information about timed-out connections or if the reported response times exceed given thresholds, it can take some further action to identify the problem. In our example it does a ping to the local router, to the PoP and to the server that the user tried to contact. The results can be used to determine if the problem is located in the dealer's LAN or in the service area of the customer. Both would mean that the provider cannot be held responsible for the malfunction. Through an ARM instrumentation of the servers and a correlation of server transactions with client transactions even more accurate information can be collected. Lynx was chosen as an example because it is a simple browser and the source code is available, so an instrumentation could be done in the course of the project. Of course, typical dealers will not be using Lynx but as ARM is gaining more and more momentum many browsers might come instrumented in the near future.

## 5 Evaluation of JDMK and ARM

This section evaluates JDMK and ARM with respect to their applicability to large enterprise management environments. We will thereby focus on the major problem domains which are of critical importance for a successful deployment and address the requirements identified in section 2.1.

### 5.1 Rapid Prototyping of Flexible, Dynamic Management Systems

As outlined in the previous sections, the primary strength of JDMK is its capability of realizing flexible and highly distributed management systems. Agents developed with JDMK can be enhanced and modified at runtime, thus yielding the opportunity of delegating management tasks to them via the push model. Furthermore, these agents can also initiate the download of management functionality by themselves (pull model). These additional services are only transferred to the agent if needed. Under regular conditions, particular tasks of the management system may be carried out by the agent, thus preventing the exchange of large amounts of data between the managing system and the agent. The services provided by JDMK enable distribution and update mechanisms to enhance agents with additional functionality residing in centralized code repositories. 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. Simple web-based user interfaces can be generated automatically by using the HTML-Adaptor. JDMK-administered resources can be easily 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 protocols not yet supported. 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.

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

### 5.2 Scalability

JDMK does neither provide standardized directory nor naming services. 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. 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. The establishment 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.

The conceptual weaknesses mentioned above impede the development of management applications for large IT infrastructures where a high number of different JDMK-based agents is needed. The JDMK services are useful for small, local environments where the amount and the 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; 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 [24] are not supported yet. 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.

### 5.4 ARM

The ARM API offers many benefits to a manager who needs to monitor the response times of applications: The most important factor is its openness. Resulting from a joint initiative of 15 companies, it offers a well defined interface that can be used either by application developers or by management tool vendors. Applications instrumented with the API calls will work seamlessly with management tools from various manufacturers (e.g. *Tivoli Application Performance Management (TAPM)*[5] or *HP MeasureWare/PerfView*[6]). However, it is also general enough to allow sufficient differentiation between management solutions from different vendors. Its adoption as a standard by the Open Group further strengthens its position as the predominant standard for response time measurement. Another important fact is its simplicity and efficiency. There are not more than six calls to be used. Technically it is very easy to instrument an application using these calls and it is also relatively easy to provide a measurement agent that implements the calls. Depending on the amount of processing the measurement agent does, it affects system performance only to a minimal level. A further benefit is the way transactions are defined. Even in a highly distributed environment it is easy to monitor the transactions a user is aware of. Users are not interested in certain parts of the transactions but in the total amount of time from their request to the reception of the response. Through the concept of subtransactions, managers still have the chance to find out which part of the transaction is responsible when performance problems occur.

However, the ARM API still comes with a lot of problems: Obviously, the first to mention is the need for instrumented applications. Today most commercially available applications are not instrumented. As normally no source code is available it is also impossible to instrument these applications on your own. Even if the source code is provided it is a difficult and time consuming task because the business transactions seen by the user must be identified in the code, which requires expert knowledge of the implementation. When using transaction correlation, the correlator received by the measurement agent must be known to the subtransactions. In a distributed environment this requires changes to the applications because the correlators have to be passed explicitly as parameters when calling the server component.

## 6 Java Management Extensions

In the middle of 1999 the draft version of the new Java Management Extensions (JMX) specification [20] 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. As JDMK can be considered an integral part of JMX, we will provide a short overview over the most recent developments and address the question whether the critical issues of JDMK also apply to JMX. However, please note that the JMX specification is not yet in a final state.

[5] http://www.tivoli.com/products/index/app_per_mgt/
[6] http://www.openview.hp.com/

JMX architecture is very similar to JDMK as depicted in figure 2. JMX distinguishes between manager, agent and instrumentation levels: Within an agent, M–Beans responsible for making a resource manageable 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. JMX MBeans[7] have been categorized into four distinct classes: standard MBean, dynamic MBeans, open MBean and model MBean. 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) [3] 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. For 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 [22] and CIM/WBEM APIs [21] are defined and implemented. The final JMX specification will provide both a reference implementation and a compatibility test suite. However, only an early access version of the reference implementation is available yet.

Based on the current draft of the JMX specification, we have to state that shortcomings of JDMK 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 has to remain an open question. This particularly applies to the manager side of JMX, which is yet unspecified.

## 7 Conclusion and Outlook

The paper described a case study for the dynamic management and monitoring of QoS parameters and SLAs based on JDMK and ARM. We have discussed our implementation concept with flexible and extensible agents which operate at the customer site. 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 allowed an 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. JDMK is best described as a development framework for Java-based *Management by Delegation*. At its current stage, JDMK is a powerful toolkit for the development of management agents that can be accessed and modified through several different communication mechanisms.

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

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 a "global" interface repository and naming 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.

The experiences of the project further allowed an evaluation of how ARM can be used for the monitoring of service levels of client/server applications. As the benefits of ARM outweigh the disadvantages by far there is hope that it will increasingly be adopted by vendors. Recently the CMG has released a preliminary version of ARM 3.0 SDK for public review (for a short overview see [9]). The most important topic of this release is the new Java binding. With this feature Java programs can use the ARM API directly and the indirection via Java Native Interface is not necessary any longer. However, until now the formal specification of ARM 3.0 is not available. Growing customer demand for applications ready for management will lead to a growing number of instrumented applications. A number of management tool vendors already offer solutions that implement the ARM API.


## 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. Application Response Measurement (ARM) API . Technical Standard C807, The Open Group, July 1998.
3. Common Information Model (CIM) Specification Version 2.2. Specification, June 1999.
4. M. Feridun, W. Kasteleijn, and J. Krause. Distributed Management with Mobile Components. In M. Sloman, S. Mazumdar, and E. Lupo, editors, *Integrated Network Management VI (IM'99)*, pages 857–870, Boston, MA, May 1999. IEEE Publishing.

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. H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application.* Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999. 651 p.
8. M. Hojnacki. Einsatz des Java Dynamic Management Kit (JDMK) zur Antwortzeitüberwachung bei der DeTeSystem . Master's thesis, Technische Universität München, May 1999.
9. M. W. Johnson. ARM 3.0 — Enabling Wider Use of ARM in the Enterprise. In *Computer Measurement Group's 1999 International Conference (CMG99)*, Reno, Nevada, USA, December, 5–10 1999. http://www.cmg.org/regions/cmgarmw/arm30enhancements.pdf.
10. K. McCloghrie and M. T. Rose. RFC 1213: Management information base for network management of TCP/IP-based internets:MIB-II. RFC, IETF, March 1991.
11. M.-A. Mountzia. *Flexible Agents in Integrated Network and Systems Management.* Dissertation, Technische Universität München, December 1997.
12. A. Pham and A. 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. J. Schönwälder. Network Management by Delegation - From Research Prototypes Towards Standards. In *8th Joint European Networking Conf. (JENC8)*, Edinburgh, May 1997.
15. Jürgen Schönwälder. *Netzwerkmanagement mit programmierbaren, kooperierenden Agenten.* PhD thesis, Technische Universität Braunschweig, March 1996.
16. R. Sturm and W. Bumpus. *Foundations of Application Management.* Wiley Computer Publishing, 1998.
17. Sun Microsystems, Inc. JavaBeans, Version 1.01. Technical report, Sun Microsystems, Inc., Palo Alto, CA, July 1997. http://www.javasoft.com/beans/docs/spec.html.
18. Sun Microsystems, Inc. Java Dynamic Management Kit – A Whitepaper. Technical report, Sun Microsystems, Inc., Palo Alto, CA, 1998. http://www.sun.com/software/java-dynamic/wp-jdmk/.
19. Sun Microsystems, Inc. Java Dynamic Managment Kit 3.0 (beta) – Programming Guide. Technical report, Sun Microsystems, Inc., Palo Alto, CA, August 1998.
20. Sun Microsystems, Inc. Java Management Extensions Instrumentation and Agent Specification, v1.0. Technical report, Sun Microsystems, Inc., Palo Alto, CA, December 1999.
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 SNMP Manager API. Preliminary Specification Draft 1.9, Sun Microsystems, Inc., Palo Alto, CA, June 1999.
23. Giovanni Vigna, editor. *Mobile Agents and Security*, number 1419 in LNCS, Berlin, Heidelberg, 1998. Springer.
24. B. Wijnen, R. Presuhn, and K. McCloghrie. RFC 2275: View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). RFC, IETF, January 1998.