Monitoring of Service Level Agreements with flexible and extensible Agents

Rainer Hauck Helmut Reiser

Munich Network Management Team
University of Munich, Dept. of CS
Oettingenstr. 67
D-80538 Munich, Germany
Phone: +49-89-2178-{2155|2163}, Fax: -2262
E-Mail: {hauck|reiser}@informatik.uni-muenchen.de

Abstract

In a customer/provider relationship a provider offers services to users who pay 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 the customer site. 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 (ARM) API. Based on our experiences with the prototype implementation this paper analyses the suitability of JDMK and ARM for the monitoring of SLAs and for building scalable and flexible management systems in large–scale enterprise networks.

Keywords:
Service Level Management, JDMK, ARM, Distributed Management, Management by Delegation

1 Introduction

In recent years some concepts have been developed to overcome the deficiencies of centralized management architectures. Examples are the Management by Delegation (MbD) paradigm, flexible agents and mobile agent technologies to build a flexible distributed management system. In such a system it is possible to delegate management functionality from the manager to the agents and there should be the ability to extend agents at runtime. Besides other features these "new" agents should have a higher degree of autonomy to act and react.

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)**. To proof the fulfillment of those SLAs it would be very helpful for the provider to have a flexible and extensible agent at the customer site which can monitor service usage and can even trigger active tests. The fact that an agent belonging to the provider is running at the site of the customer implies strong security requirements.

The paper shows how the concepts and services of the Java Dynamic Management Kit (JDMK) can be used for building such an architecture. It also shows weaknesses and deficiencies JDMK still has to deal with. As an example of how the 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 real QoS parameters can be obtained is shown. Section 5 summarizes our experiences and evaluates how JDMK and ARM are suitable for monitoring SLAs in large enterprise environments.

2 Scenario: Outsourcing of Extranets

An exemplary scenario – which this paper is based on – is a car manufacturer (customer) which enables all

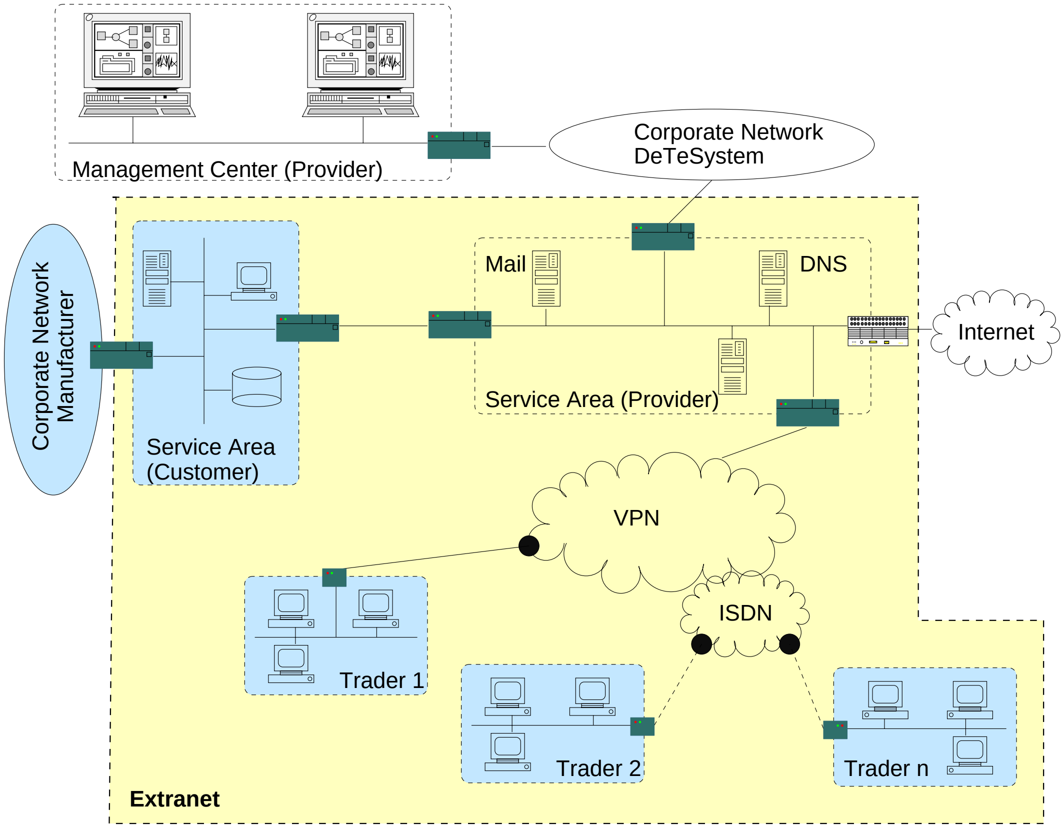

Figure 1: Extranet Solution for a Car Manufacturer

its traders (more than 1000) to use special applications within the corporate network (CN) of the manufacturer (e.g. Online Ordering of cars). The traders build a so called extranet (EN). Besides the specific applications the EN enables connections to the Internet. The provider side is represented by DeTeSystem, a network and system provider and outsourcing partner for the biggest customers of Deutsche Telekom AG. 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 which are offered by itself. This means that the provider is not responsible for applications offered by the customer.

The corresponding infrastructure is represented in figure 1. Traders 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 trader wants to use a service of the manufacturer it will be routed into the SA of the car manufacturer. In this configuration traders use services in the SA of the provider as well as in the SA of the customer.

2.1 Management Requirements for Monitoring SLAs

In order to observe the quality of delivered services it is necessary to negotiate **Quality of Service (QoS)** parameters as well as modes for the measurement and evaluation of these parameters. QoS parameters must reflect the expectations of the customer. They are parts of Service Level Agreements between the customer and the 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 the QoS parameters have to be supervised by the management system of the provider. Further the provider is obliged by the SLA to report the compliance with agreed QoS parameters.

An example for such QoS parameters are the response time, the connectivity or the availability of a certain service. It must be remarked that these parameters have to be measured and valued from the customer point of view. Most of the QoS parameters (e.g. the connectivity or the availability) of a service are only defined if the customer tries to use this certain service. For the monitoring of these parameters it is necessary for the management system of the provider to be able to measure from the side of the customer, i.e. that in our scenario an agent which performs the monitoring of QoS parameters must be installed at a trader's host.

A customer will allow the provider to install an agent at it's systems only if it meets high security requirements and if no additional or only minimal costs are caused by the agent. This means that such an agent must not initiate connections — especially for customers which are connected via a dial-up line. The agent has to do its measures locally as far as possible and transfered results should be as small as possible.

Requirements for services, QoS parameters and SLAs could change in course of time. Therefore it is necessary that the architecture is flexible and it is able to react on such changes quickly. It must be possible to extend the agent with additional functionality, e.g. for the monitoring of a new QoS parameter.

In such large–scale corporate networks with thousands of hosts and locations all over Germany and Europe it is absolutely essential that the management system scales very well. 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) [4, 14, 15] 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, pre-defined tasks. However, the applicability of mobile agents is bound by security concerns; [5] and [20] 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, [3] presents a Java-based environment for configurable and downloadable lightweight management applications.

Taking care of the growing importance of customer care is the aim of **Customer Service Management (CSM)** [9]. It offers a management interface between customer and service provider, which enables the customers to monitor and control their subscribed services.

3 Framework and Development Tools

The Java Dynamic Management Kit (JDMK)[1], developed by Sun Microsystems [18, 19], promises to overcome many of the mentioned problems 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 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 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.

M–Beans (Managed Beans) are Java objects implementing the intelligence and the functionality of an agent. An M–Bean acts as a representative for one or more **managed objects (MOs)**. 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 the same as the name of the class implementing the M–Bean. The optional

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

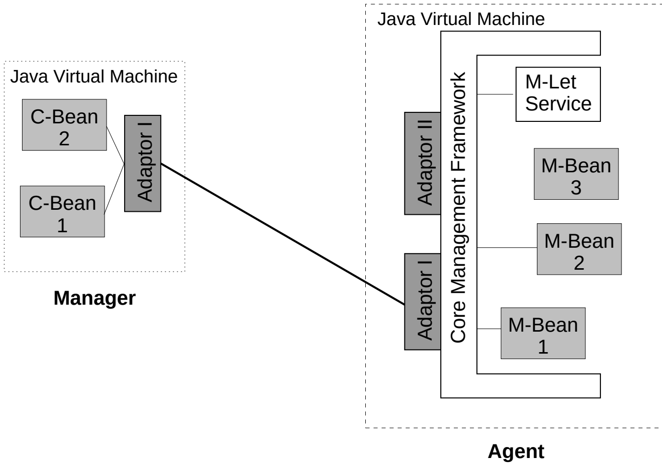

Figure 2: JDMK Architecture

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).

In order to use JDMK services or communication resources the 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 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 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`). C–Beans 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 hence 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. Of course it is possible to use more than one adaptor at the same time.

3.2 JDMK Services and Development Tools

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

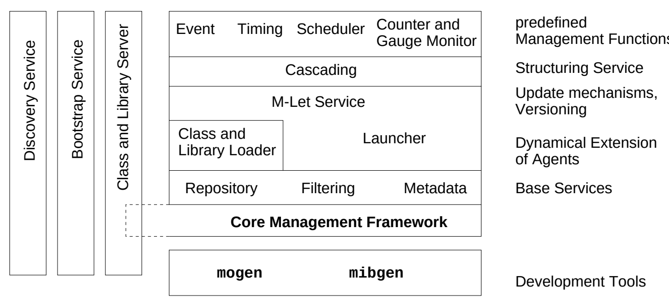

Figure 3: JDMK Services and Development Tools

Repository Service is used for the registration process. 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.

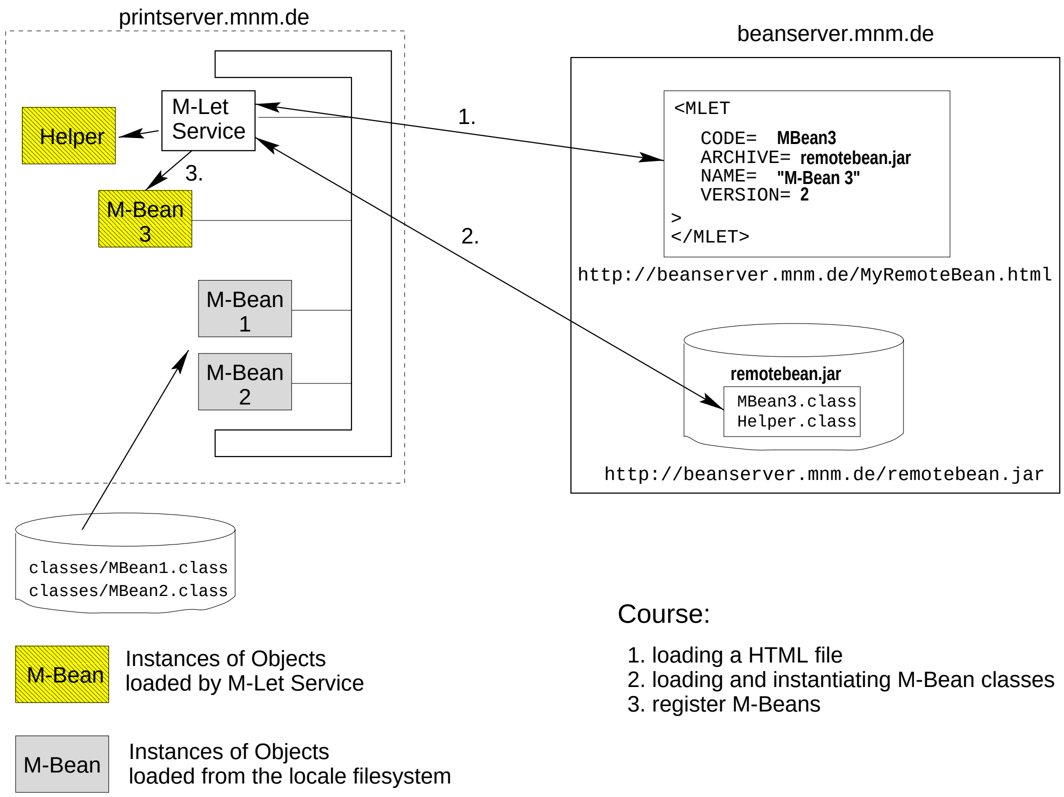

Figure 4: The JDMK Management Applet (M-Let) Service

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. The **Class and Library Server** is available as a local or remote repository for class files and libraries. This server can be used either as a stand-alone application or – because 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 with 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 implemented as a stand-alone Java application, simplifies the distribution and instantiation of JDMK agents. This 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.

Two compilers, `mogen` and `mibgen`, are delivered with JDMK as development tools. The `mogen` is used to create C–Beans from M–Bean `.class` files.
For developing an JDMK based SNMP Agent for a device `mibgen` can be used. If SNMP–MIB files are available for the managed 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.

3.3 The Application Response Measurement API (ARM)

In today's distributed computing environments traditional approaches to response time monitoring no longer seem to be adequate. Host centric approaches face the problem of not taking into account the distributed nature of the application to monitor. Network based approaches in contrast often are unable to map the packets found on the network to transactions meaningful to the user. Moreover these solutions are frequently proprietary and do not allow integration of applications and management tools from different vendors.

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)[2] to continue development and promotion of the ARM. 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.

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 are instrumented to call the following API functions:

- `arm_init`
  Is called during initialization of the application to be monitored and names the application and the current user.
- `arm_getid`
  This call is used to define the various classes of transactions appearing throughout the application.
- `arm_start`
  A call to `arm_start` is inserted into the application's code just before a transaction to be measured begins. Data to describe the current status of the application can be included.
- `arm_stop`
  This signals the end of a transaction. As seen with `arm_start`, data can be included.
- `arm_update`
  To provide status information about a running transaction, `arm_update` is used. It can be called any number of times between an `arm_start` and an `arm_stop`. It can also be used as some kind of a "heartbeat" signal to show that a long running transaction is still alive.
- `arm_end`
  Cleans up the ARM environment. A call to `arm_end` should be made when the application is ended or if no further monitoring is required.

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

A library implementing the calls has to be linked against the instrumented applications. The ARM Service Development Kit (ARM SDK) provides two libraries. On the one hand there is the so called NULL library, which simply returns TRUE for all the API calls. It can be linked against an instrumented application when no monitoring is to be done and it effects system performance only to a minimal extent. On the other hand there exists a library which implements a logging agent, that means an agent that writes all the information received by the application to a file. The logging agent is considered to be used mainly for testing purposes.

Actual performance monitoring can be done by using management tools that provide a measurement agent that implements the API and that is also linked against the application to be monitored (see figure 5).

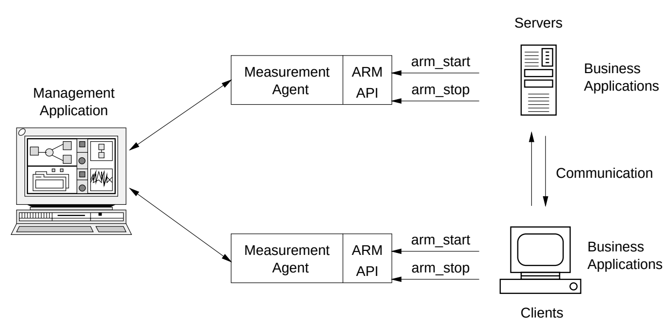

Figure 5: Using the ARM API

An important feature added with version 2 of the API is correlation of transactions. Often one transaction visible to the user consists of a number of subtransactions. When indicating the start of a new transaction using `arm_start`, a data structure called a correlator can be requested from the measurement agent. When indicating the start of a subtransaction (again using `arm_start`), this correlator can be used to inform the measurement agent about the existence and identity of a parent transaction.

4 Monitoring of Service Level Agreements using JDMK and ARM

In an industrial cooperation project with DeTeSystem [7] a prototypical management solution was developed that fulfills the requirements mentioned above. In this project JDMK (version 3.0 beta 2) and ARM (version 2.0) were used for the monitoring of Service Level Agreements.

4.1 Architecture

As seen before there is a strong requirement for Service Level Monitoring to be seen from a customer's perspective. QoS parameters of SLAs are only reasonable if the user actually tries to use a service. However, if it does not try to use a special service there could not be a violation of a 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 disconnecting of the link if it's 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 today's ISDN routers, as there is no way to distinguish between packets that should affect the idle timer and packets that don't. As a trade-off the following solution was chosen: Management traffic is created only when the link is up. Although being far from optimal 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 agents 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: The agent can poll the variable `ifOperStatus` (operational state of the interface) from MIB-II [10] on a regularly basis. As a relatively high polling interval can be chosen (about half of the idle timeout value for the link) the additional load caused is minimal. That means that seen from the perspective of the router, the agent acts in the role of a SNMP manager.

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 a 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 trader from explicitly deleting the information collected by the agent before it is transmitted to the manager.

Another benefit of using the JDMK was it's 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 easily be achieved. A manager application was build 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 Service Level 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 the web servers in the service areas of both the provider and the customer. The transactions of interest are the download of certain web pages by the traders. 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 6 shows the transactions to be monitored.

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

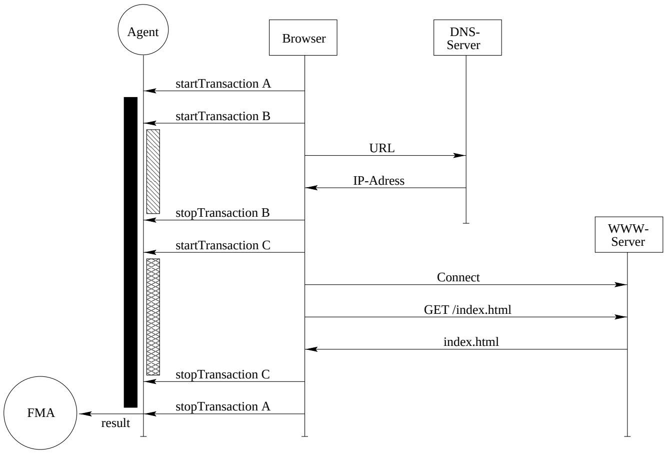

Figure 6: ARM instrumentation of a web client [7]

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 trader's LAN or at a server 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 traders 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 its 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 strengths 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 themselves the download of management functionality (pull model). These additional services are only transferred to the agent if needed. Under regular conditions, particular tasks of the management system 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 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, 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.

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 Scalability

JDMK does neither provide standardized directory and 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 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.

The conceptual weaknesses mentioned above 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 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 [21] 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.

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. However, it is also general enough to allow sufficient differentiation between management solution from different vendors. Its adoption as a standard by the Open Group further strengthens its position as the one 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 its 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. The NULL library allows applications to run as if they were not instrumented at all.

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 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.

Another problem is the use of fixed data structures. As mentioned above information about the currently running transaction can be included in the API calls. However, the type of data is predefined to a maximum of six numerical values and a string of length 32. In our implementation this caused a problem because it was necessary to include the URL of the requested page into the packet, which often exceeds a length of 32 characters. By using `arm_update` this problem could be avoided, because it allows to include up to 1020 bytes of unstructured data.

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 Conclusion and Outlook

This paper describes a case study for the dynamic management and monitoring of 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 allow 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 (Version 3.0 beta 2), JDMK is a powerful toolkit for the development of management agents that can be accessed and modified through several different communication mechanisms (RMI, HTTP, SNMP, HTTPS, CORBA/IIOP).

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.

JDMK is not positioned as a stand-alone management framework but serves as the communication infrastructure of Java Management API (JMAPI)[3] version 2, Sun Microsystems' emerging Java-based management framework. Consequently, the Sun management system Solstice will make extensive use of JDMK. It is therefore expected that the future development of JDMK will eliminate the current weaknesses.

The experiences of the project further allow an evaluation of how ARM can be used for the monitoring of service levels of client/server applications. Section 5.4 mentioned pros and cons of the ARM API. As the benefits of ARM outweigh the disadvantages by far there is hope that it will increasingly be adopted by vendors.

Currently the ARM Working Group is developing version 2.1 of the API [8] which e.g. will address the following topics:

- Allow applications to provide unique transaction identifiers to the agent. This is especially important when instrumenting server subsystems and middleware components which often already have their own unique IDs for transactions.
- Define an API call for legacy applications which already provide some kind of performance monitoring. An `arm_complete_transaction` call will be defined that enables an agent to receive information about a complete transaction.
- An object oriented version will be provided that uses the benefits of object orientation while still permitting full interoperability with other (non-OO) ARM implementations.
- A solution for the monitoring of Java applets will be provided. Applets face the problem that they cannot call local libraries, so the measurement agent has to be included into the applet.

Growing customer demand for applications ready for management will lead to a growing number of instrumented applications. A number of management

[3] http://www.javasoft.com/products/JavaManagement/

tool vendors already offer solutions that implement the ARM API. The Desktop Management Task Force (DMTF) in 1998 formed a new subgroup called "Distributed Application Performance"[4]. It is developing a model for application performance that should be consistent with other CIM models and with the ARM API. This will further improve acceptance of the ARM as the standard for application response time measurement.

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, TOG, July 1998.

[3] 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.

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

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

[6] H.-G. Hegering, S. Abeck, and B. Neumair. *Integriertes Management vernetzter Systeme — Konzepte, Architekturen und deren betrieblicher Einsatz.* dpunkt-Verlag, 1999.

[7] M. Hojnacki. Einsatz des Java Dynamic Management Kit (JDMK) zur Antwortzeitüberwachung bei der DeTeSystem . Master's thesis, Technische Universität München, May 1999.

[8] M. W. Johnson and S. Smead. Beyond ARM 2.0 – API Extensions that Enable Pervasive Service Level Instrumentation. Technical report, Computer Measurement Group (CMG), December 1998. `http://www.cmg.org/regions/cmgarmw/arm21/index.html`.

[9] M. Langer, S. Loidl, and M. Nerb. Customer Service Management: A More Transparent View To Your Subscribed Services. In A. S. Sethi, editor, *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, Newark, DE, USA, October 1998.

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

[21] 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.

[4] http://www.dmtf.org/info/dap.html