Proceedings of the *Seventh IEEE/IFIP Network Operations and Management Symposium (NOMS 2000)*, Honolulu, HI, USA, April 2000

Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis

G. Kar, A. Keller, S. Calo
IBM T. J. Watson Research Center
30 Saw Mill River Road,
Hawthorne, NY 10532, USA
{gkar|alexk|scalo}@us.ibm.com

Abstract

This paper proposes a novel approach for managing IP-based services and applications, reflecting the authors' experience with the IBM Global Network. It describes how one can *extend* the existing network and systems management paradigms to address problems in the management of application services hosted by Network Service Providers (NSP). We introduce the concept of "service management domains" which are virtual domains built from resources and dependency relationships pertaining to physically monitored domains in the network management layer. They form the basis for managing fault, performance and service level agreements related to application service offerings.

Keywords

Application Management, Dependency Analysis, Automated Problem Determination

1. Introduction

Throughout this decade, packaged connectivity services have mushroomed into a multi billion dollar market. Network Service Providers (NSP), also referred to as Internet Service Providers (ISP), sell network connectivity to corporations with geographically dispersed locations. An NSP, such as the IBM Global Network, defines a virtual private network over its wide area network infrastructure for this customer so that the different locations can exchange information. Customers are eager to have such a service since their connectivity costs are considerably lower than leased lines and a great deal more flexible in terms of bandwidth assignment. The NSP's benefits stem from providing connectivity services for many customers on their shared physical infrastructure. Since the mid 90's, however, as bandwidth has increasingly become a commodity, and – as more and more NSPs have entered the market – the profit margin for such connectivity services has gone down. NSPs have gradually moved beyond the network layer to deploy and offer shared **application services** [17]; consequently, they are considered Application Service Providers (ASP). Examples are: content hosting for data with web-based access, shared payroll applications, firewall-based security services, email and shared file services, etc. To offer these value added services in a cost effective manner, NSPs set up and maintain large server farms to host applications. The customer is offered – in addition to basic connectivity – access to "end-to-end" managed application services, associated with service level agreements (SLA) [20].

From the NSPs' point of view, application service provisioning has brought a new challenge, namely how to manage these services so that high availability is guaranteed. In general, NSPs already have well-developed network management infrastructures to operate their physical networks consisting of servers, switches, routers, links, etc. Thus, a very important objective is to leverage the existing network management infrastructure and enhance it to provide application service management [9, 18]. Research [8] has shown that a critical aspect of problem determination in a complex environment is the identification and representation of *dependencies*, i.e., the relationships that exist between elements of various layers of a distributed system. This paper describes an architecture that addresses the above points by extending the existing management infrastructure with application service management capabilities.

Section 2 gives an overview of a typical service provider network and its management architecture. Section 3 introduces the notion of application services and defines the concept of service management domains which comprise collections of resources that interoperate to provide a service. Within these service management domains, dependencies exist between different kinds of resources. A repository-based methodology for determining and analysing these dependencies is presented in section 4. The generated dependencies are used as input for our application service management architecture whose elements are described in section 5. Section 6 concludes the paper and presents issues for further research.

2. NSPs: Moving from mere Bandwidth to value-added Services

The IBM Global Network has points of presence in over 800 cities and 100 countries; it offers IP and SNA application and connectivity services defined over Frame Relay and ATM. In this networked environment, a typical scenario for networked application services consists of three distinct entities: the customer premises environment (CPE), the wide area network and an application hosting farm.

In traditional network management, the general approach is to partition a physical network environment into what is known as **monitored domains**, each of which is under the supervision of a (typically SNMP-based) management system. They are defined according to topological considerations. An application service, comprising instances of distributed components, such as database servers, domain name servers, various middleware etc. across such a network, depends on the performance and status of resources that belong to multiple such monitored domains. Therefore, monitoring the health and status of an application service requires coordinated monitoring and collection of selected information across multiple physical domains. This paper introduces the concept of **service management domains** which are virtual domains built from resources and relationships pertaining to the monitored physical domains. They form the basis of information monitoring for managing fault, performance and service level agreements related to application service offerings.

For a large network with many monitored domains, centralized management generates a prohibitive amount of polled data and asynchronous event traffic, a large portion of which is generally useless for operational (i.e., fault and performance) management.

Therefore, it is common practice to implement hierarchical management with mid-level managers (MLM), each in charge of a set of domains, as shown by the dotted boundaries in Figure 1. Standard network management platforms, such as Tivoli TME, HP OpenView and Cabletron Spectrum provide for such a hierarchical, distributed architecture [7].

Mid-level managers can be thought of as dual-role entities (agent and manager) that process raw data into meaningful management information. From an architectural view point they appear as extended SNMP agents to the top-level Network Operations Center (NOC) management station. The functions performed by a MLM are:

- polling the SNMP agents in its domain; variables being polled and the polling frequency are determined by a policy that is periodically configured on the MLM, to reflect changes in topology.
- reception of events/traps from the SNMP agents; event filtering based on rules that are defined by the top-level manager and downloaded to the MLM.
- storage of polled data locally in a database. This stored data (or a summary of it) is periodically sent up to the top-level manager(s) using some bulk data transfer protocol such as FTP. This data is then analyzed by applications located at the top-level manager to compute SLA statistics, produce histograms, create trending reports, etc.
- filtering and forwarding of selected events/traps to the top-level manager and to its fault management applications, respectively. In our scenario, such events are received (with appropriate transformation done by adapters) by the Tivoli Enterprise Console (TEC) which allows a network manager to specify rule-based, event-triggered automation routines for efficient fault detection and resolution.
- run a backup protocol so that if one of the MLMs goes down, another can pick up its responsibilities until the former is available again. In our environment, this fault tolerance and availability mechanism has been implemented in a way where each MLM has a designated backup.

Today, most large data networks have an operational management system that can be represented by this model. As more and more network service providers offer managed application services in addition to mere IP connectivity, it becomes important to understand how such an existing network management infrastructure can be enhanced to provide management functions for application services. As we will analyse in the next section, this transition brings forth important requirements.

3. Requirements for Application Service Management

In general, application services are provided by server farms either owned and operated by the service provider itself or by a third party. Every customer of a managed application service expects to receive prompt notification and corrective action from the service provider when his applications are affected by faults or performance problems. The extent and quality of this corrective action is specified in SLAs [4] between the customer and the service provider. In order to provide end-to-end application service management for every customer, the service provider has to deploy a service management system in addition to the operational network management system in order to be able to provide the following key functions:

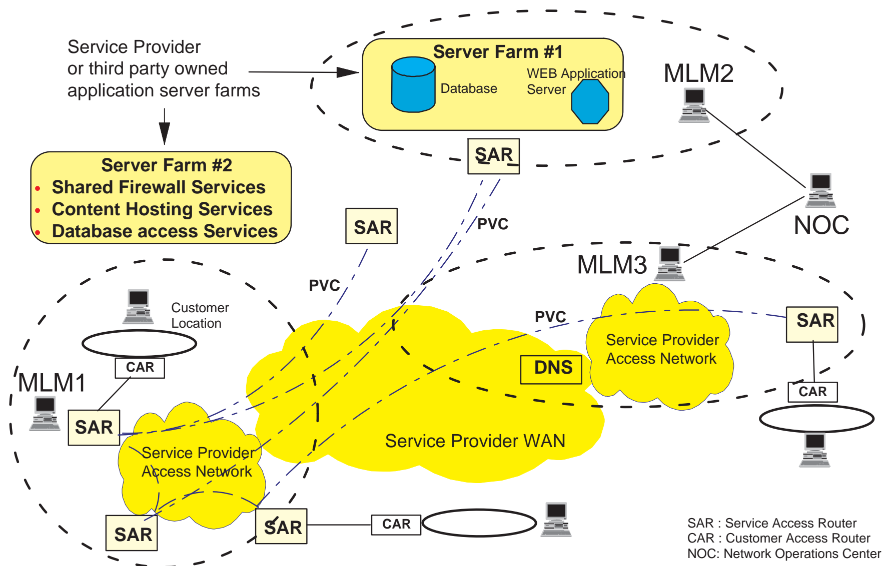

Figure 1: Architecture of Application Services over Networks

- **Root Cause Analysis**: Depict a particular customers' service status, much the same way as one is used to seeing the status of a network resource. This graphical interface should provide a "drill down" facility such that, when a problem is indicated, the operator is able to traverse the layers down to the network-level resources on which the service depends. Ideally, the NSPs' management system should be able to pinpoint the cause of the problem and fix it.
- **Impact Analysis**: When a resource goes down, determine (in realtime) which services and customers are affected. The information obtained by this analysis is also particularly useful for scheduling maintenance tasks: It is entered into a customer relationship management (CRM) system to issue proactive warnings to targeted customers.

These requirements are best illustrated by the example shown in Figure 1, depicting the essential constituents of a service defined over a service provider network: Three customer locations are connected to each other by means of frame relay permanent virtual circuits (PVC). Two server farms, maintained and administered by the service provider, offer a variety of application services to the customer. For example, customers may subscribe to the web content hosting service, whereby their on-line product catalog is maintained by the service provider. Such a service may provide multiple functions: customer-initiated content creation and storage, public and restricted access to the customers' content over the Internet, statistical reports on the usage of the data, etc. The data may be spread across the different customer locations, thus introducing two distinct classes of dependencies:

Intra-System Dependencies occur *within a single system*, i.e., these dependencies denote whether a specific service requires another service that is installed on the same system. An example for this kind of dependency is a relationship indicating that a WWW

service depends on the availability of the Domain Name System (DNS) which, in turn, requires a functioning IP service.

Inter-System Dependencies are dependencies between services located on *different systems* and have key importance for end-to-end problem determination. Typical Inter-system dependencies exist between the client and server parts of a given service, e.g., a Network File System (NFS) client is only able to perform its work only if its peer NFS servers are operational.

However, current management systems are unable to deal with service dependencies because topological domains for network management differ largely from service management domains w.r.t the number and dynamics of managed objects. A common workaround used by NSPs is to define a view for *every* customer containing all the resources the subscribed services depend on (PVCs, network components, servers, applications). Although this yields a customer-focused view on applications, services and networks, it has significant drawbacks:

- Resources are assigned to views manually by the operators and not automatically by the management system. This implies that these views do not reflect the dynamic behaviour of networks and services.
- The manual placement of icons in views is unscalable, error-prone and tends to yield inconsistent data.
- As motivated in section 1, the main benefit for NSPs stems from the fact that resources (such as routers, servers, firewalls etc.) are shared between different customers, thus maximizing the resource usage. Introducing customer-focused views on the management system implies that an icon representing a shared resource appears multiple times in different views, thus yielding redundancy.

In order to avoid these drawbacks while being able to provide the above listed service management functions, we need a methodology that allows us to determine:

- a dependency model that represents the relationships these resources have on each other, e.g., the DNS and NFS availability is dependent on the IP service being available from the customer environment, which in turn is dependent on the proper functioning of the customer and service access routers (CAR and SAR) and the PVCs. Such a dependency model will allow a fault management application to perform root cause analysis of a problem perceived by an application service user. In general, constructing such a dependency model is very difficult (see e.g., [8]). Our approach, presented in section 4, is based on a static dependency analysis.
- a set of all resources in the environment that affect the service either directly or indirectly. In our example, these would be the CARs, SARs and the PVCs that provide the IP connectivity between a customer location and the server farms, the NFS and DNS and the database systems in the server farm. Our architecture, discussed in section 5, defines a virtual **service management domain** on these resources under the control of a service management agent.

Resources contributing to the functioning of an application service belong to different physical domains, each under the monitoring scope of a MLM. In Figure 1, MLM1, MLM2 and MLM3 monitor the resources that need to be operable for the content hosting service to be functional. A key requirement for our service management architecture is

to be able to logically relate a service management agent to the relevant MLMs, so that critical status information can be delivered to it. As discussed in section 5, our architecture uses the concept of a **resource broker** in conjunction with a **resource directory** to address this issue.

4. Dependency Analysis: A pragmatic Repository-based Approach

The analysis in the previous section has pointed out that a major requirement for the automated management of distributed application services is to have a record of their dependencies on lower layer services and resources. Abstract service models have been used to address this issue [14], but this approach requires the collection and mining of large amounts of data. In the following sections, we will describe a pragmatic approach for dependency enumeration that can be realized on a variety of widely deployed operating systems *without* requiring detailed management instrumentation of the applications and networked services.

Furthermore, despite early work [19] and several research [5] and standardization efforts [15, 12, 6] no satisfactory application service management solutions are available on the marketplace. The main problem is that comprehensive application management demands a large amount of management information, thus posing an additional development effort on the application developers [16, 13]. A good example for this is the *Application Management Specification (AMS)* [1] that provides an open standard for defining the management information needed for distributed applications. While AMS identifies a set of management information common to different kinds of applications and the means of specifying it using application description files, the application developer needs to provide the appropriate instrumentation.

4.1 System Information Repositories

Considering the fact that a majority of application services run on UNIX and Windows NT-based systems, it is worth analyzing the degree to which information regarding applications and services is already contained in the operating systems. The underlying idea is as follows: if it is possible to obtain a reasonable amount of information from these sources, the need for application-specific instrumentation can be greatly reduced. Our approach recognizes the fact that system administrators successfully deploy applications and services without having access to detailed, application-specific management instrumentation.

Windows NT/95/98 systems and UNIX implementations such as IBM AIX and Linux have built-in repositories that keep track of the installed software packages, filesets and their versions. AIX *Object Data Manager (ODM)*, Windows *Registry*, and Linux *Red Hat Package Manager (RPM)* are examples for these system-wide configuration repositories. In this paper, we will concentrate on ODM. However, our approach is applicable to other repositories as well.

Usually, these repositories serve as the basis for software installation tools such as *InstallShield* (for Windows systems) or the AIX *Systems Management Interface Tool (SMIT)*. Moreover, they can be regarded as knowledge bases that contain important information with respect to the compatibility of services and applications. The fact that a

specific software package must already be present on a system so that another package can be installed successfully, implies that the service implemented by the latter package depends on the service implemented by the former. In other words, if a specific software package has other software packages listed as installation prerequisites, we can deduce that this dependency relationship is also valid for their instantiated counterparts, i.e., services and applications.

A simple example will serve to illustrate this: if the system repository indicates that a specific Web server has a particular TCP/IP implementation as a prerequisite, this essentially means that:

1. A **functional**, i.e., generic and implementation-independent relationship model for the service categories “WWW” and “TCP/IP” can be established. The model describes services in terms of the functionality they provide and on which other services they depend. The fact that the WWW service depends (among others) on the availability of the TCP/IP service implies that a functional dependency relationship between the services ”WWW” and ”TCP/IP” is defined (by means of the description in the prerequisites list) and enforced by the installation routine.
2. **Structural**, i.e., specific and implementation-dependent management information is available. An appropriate algorithm for automated problem determination for a malfunctioning WWW service would have to check whether the operational states of a specific Web server and of its underlying TCP/IP stack are ”up”. This is possible because the dependency relationships of a specific service are explicitly listed in the system repository. Section 4.2 will give an example of such a data structure.

Discovering and enumerating the dependency relationships that applications have on lower layer services in a networked environment is a difficult problem. It has both a static and dynamic aspect, that is, dependencies identified at application install time and those discovered at runtime. The functional dependency model can be used to describe static dependencies between application and service categories. The structural part captures dynamic information related to concrete service implementations. In the next sections, we illustrate our approach by means of an example.

4.2 Intra-System Dependencies: The IBM AIX Object Data Manager

The *Object Data Manager (ODM)* is the repository of the AIX operating system and contains information about the different components of the operating system such as device drivers, networking services and graphical user interfaces, middleware (like object request brokers, message-queuing systems), development tools (compilers, debuggers, CASE-Tools) and additional components such as database and Web servers. Furthermore, applications such as Web browsers, database clients, management platforms, groupware systems are also registered in the repository.

ODM obtains knowledge about new software components at their installation time from the `installp` software installation routine that is invoked by SMIT. This routine reads the software description template (an example of this simple template common to all applications is depicted below) that comes with a new software package, verifies the prerequisites and – upon successful completion of the prerequisite tests – installs the software and enters the description template into ODM. Note that the information

contained in the template can be added after the application has been compiled. Thus, it does not necessarily have to be provided by the application developer (although this is desirable) but can be added by the system administrator or a third party.

We will now take a look at how configuration information for software packages is represented in ODM. The following entry for the successfully (`state = 5`) installed TCP/IP client component* (`bos.net.tcp.server`) – being part of the operating system networking package (`bos.net`) of IBM AIX – indicates that this software (version `4.2.1.0`) replaces and renames the previously installed component `bosnet.tcpip.obj` version `4.1.0.0`.

```
lpp_name = "bos.net.tcp.server"          /* component name            */
comp_id = ""                             /* component identifier      */
update = 0                               /* update (yes/no)           */
name = "bos.net"                         /* package name              */
state = 5                                /* committed/applied/rejected*/
ver = 4                                  /* component version#        */
rel = 2                                  /* component release#        */
mod = 1                                  /* component modification#   */
fix = 0                                  /* fix (yes/no)              */
ptf = ""                                 /* fix id                    */
media = 3                                /* install media type        */
sceded_by = ""                           /* superseded by             */
fixinfo = ""                             /* verbose fix description   */
prereq = "*prereq bos.rte 4.2.0.0, *prereq bos.net.tcp.client 4.1.0.0"
description = "TCP/IP Server"
supersedes = "bosnet.tcpip.obj 4.1.0.0"
```

Additional information for applied fixes/patches and their descriptions is also found in this template. One particularly important part of this data structure is the entry ”`prereq`” (prerequisites) because it denotes which other software packages must already be present in the system so that this component can be successfully installed. In the following sections, we will describe how we make use of this information for our purposes. Since every software package installed on AIX is required to list its properties in this machine-readable format, we can therefore assume that the dependencies cover a comprehensive set of software packages. ODM can be accessed in three different ways:

1. The most common mechanism for system administrators to interact with ODM is SMIT, a graphical user interface that provides menus for common system administration tasks such as, installing and configuring system and application components.
2. In addition, a command line interface exists which is used for accessing ODM data from shell or Perl scripts. ODM entries can be searched according to different criteria and retrieved (`odmget` command), modified (`odmchange`), added (`odmadd`) or deleted (`odmdelete`). The ODM data structures can be displayed using `odmshow`.
3. The most elaborate way to issue various operations against ODM is through a C-API containing 22 subroutines that cover a variety of query and administrative functions. Details can be found in [10].

*The term “LPP” in the template stands for “licensed program product” and denotes a service component.

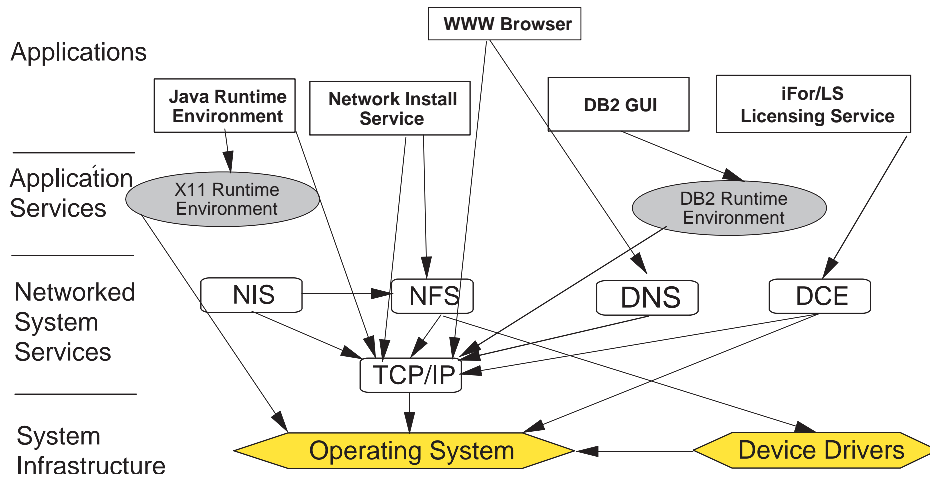

Figure 2: Automatically generated functional service dependency model (extract)

By using the query operations of the ODM API, it is possible to obtain both **functional and structural service dependency models** at runtime. Figure 2 represents an extract of the functional service dependency model (including applications such as DB2 database clients, services such as NFS and DNS) of a running system and *not an abstract model*. It is built by evaluating the `description` and `prereq` fields of the above template. The structural dependency model is much more detailed and omitted for the sake of brevity; it contains information on the different components that make up a service and is based on the `lpp_name`, `ver` and `rel` entries in addition to the both fields cited above.

This analysis shows that system repositories such as ODM represent a rich source of application management information, not only regarding the configuration of installed applications but also for determining dependency relationships between applications and services. Note that this large amount of information has been obtained *without any* specific instrumentation of the components. The only requirement is that the application components be described using the above template. However, it should be noted that:

1. system repositories refer to *individual systems only* and thus yield, for the most part, Intra-system dependencies. Since end-to-end management mainly deals with Inter-system dependencies, we have to develop mechanisms that allow us to track the dependencies between client and server components located, in general, on different systems. This topic is addressed in the following section.
2. it is necessary to link the **static** information contained in the repository to **dynamic** information obtained from the processes at runtime (e.g., their state and priority, the percentage of used CPU cycles and memory, the userID of their owner etc.). A mandatory requirement for this is that the templates also contain the process names of their components (which is not the case in the current ODM implementation). However, as the process names are already known at installation time, they can easily be included into ODM by adding an additional item “`process name`” to the data structures.

4.3 Inter-System Dependencies

As mentioned in Section 3, it is particularly important for end-to-end problem determination that information about Inter-system dependencies be available, because it is necessary to determine if the client *and* the server parts of a given service work properly. We will first examine the possibilities of determining problems on the client side and then describe what mechanisms are available for servers.

Very often, information about the servers to which a client connects is explicitly listed in the client configuration files: A DNS resolver not only has information regarding its domain name but also its name servers. NFS clients - in turn - know from which servers they need to mount file systems. Some counterexamples of this are the WWW and FTP services, where the server names are obviously not known in advance. However, this problem is alleviated by the fact that usually a proxy server is known, thus allowing one to determine if the problem is local to the client or lies beyond the proxy.

On the server side, we are able to make use of the dependencies listed in ODM also for the determination of Inter-system dependencies for the following reasons:

- Client and servers usually share the same functional dependencies (e.g., both Web client and Web server depend on the availability of the DNS service which - in turn - depends on a working IP service). The functional service dependency model in Figure 2 is valid for clients and servers.
- From an installation point of view, reflecting the dependency structure stored in ODM, clients of a given service are prerequisites for their server counterparts. This implies that server software can only be installed if the client part is already present on the system; the ODM template for a TCP/IP server described in section 4.2 clearly states this in the second argument of the `prereq` attribute. We can therefore assume that the clients needed to test the servers are locally installed thus facilitating the testing of servers because no network or intermediate systems are involved.

Although we have seen that system repositories provide a considerable amount of information useful for application management, we should point out that one specific kind of dependency cannot be tracked by them: it is yet not possible to determine if a software package is installed on a local or remote file system. The additional dependencies introduced by networked file systems (and their underlying operating systems) do not appear in the model. Appropriate problem determination algorithms have to verify whether the installation path (listed in the repository) refers to a local or a remote system (by examining e.g., the NFS configuration) and update the model accordingly.

Another issue is that the information identified in this section is relevant for configuration and fault management but not suitable for performance and accounting management: It is possible to verify whether a service and its prerequesites and peers are working but we cannot ascertain whether the services perform their duties in an acceptable time period. In order to satisfy performance-related SLAs, the applications need to be instrumented appropriately. The Tivoli *Application Response Management (ARM) API* [2] with its recent extensions [11] is a first step in this direction.

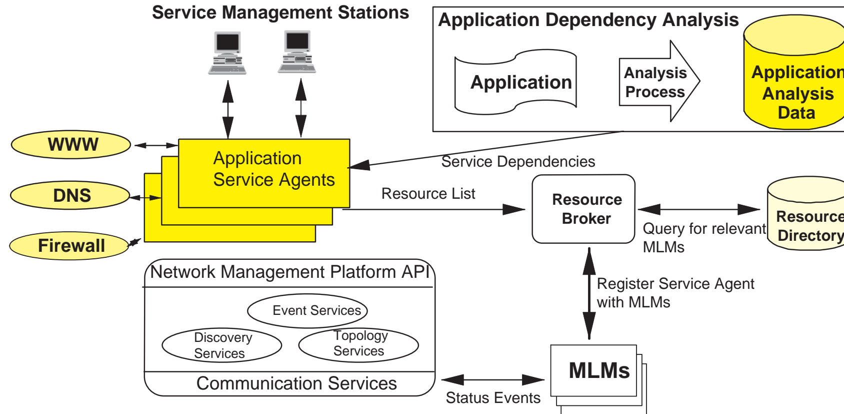

Figure 3: Components of the Application Service Management Architecture

5. An Architecture for Application Service Management

As mentioned in section 1, a major requirement is that an approach for managing application services needs to address the fact that NSPs have made significant investments in network management systems. Our overall architecture is depicted in Figure 3: Off-the-shelf network management platforms provide the basis of this architecture and offer services such as event reception and forwarding, resource discovery functions or topology services [3].

5.1 Application Dependency Analysis

As a result of the static analysis during application installation and provisioning, each application service offering has associated with it a list of resources in the network management layer that provide the basis for that service. This data is kept in a database constructed and maintained by the application dependency analysis stage discussed in the previous section and depicted in the upper right part of Figure 3. In the case of the content hosting scenario discussed in section 3 (and shown in Figure 1), the resource list associated with this application contains the resources that support the IP connectivity, the back-end database, the NFS and the DNS.

5.2 Components of the Architecture

Application Service Agent

The application service agent is the focal point for managing an individual service offering. From an implementation viewpoint, it is like an application operating on top of a network management platform. If, for instance, the NSP has three different service offerings (DNS, web-based content hosting and shared firewall) our design yields three service management agents, each responsible for one offering. The agent receives event notifications from the MLMs through the platform API and updates the view of the service that

it maintains. This view can best be represented by a multi-level resource tree, where the elements in one level are dependent on the availability and status of elements at the next lower level. One way to use the service view is to represent it graphically in one of the service management stations where a service manager can observe the status of the service and do typical drill down operations for troubleshooting.

The functional dependencies yield a generic service model while the structural part provides detailed information on the involved components. While the functional model is stored at the MLMs and the management platform, the application service agent maintains a structural view for every individual customer. This leads to a high degree of distribution and is the main reason for the scalability of our approach; thus, an outage in any resource can be rapidly correlated with complaints received from that customer. During initialization, a service agent obtains the configuration file generated at the application dependency analysis phase, which is used to establish the structural model.

Resource Broker

The function of the Resource Broker is to serve as the main access to the Resource Directory. The entities that interact with the Resource Broker are the Application Service Agents and the MLMs: An Application Service Agent sends the list of resources contained within its service view to the Broker. The MLMs communicate with the Resource Broker in order to register the resources within their domain and to update these records periodically. The Resource Broker is then able to establish associations between an Application Service Agent and a set of MLMs, thus making it possible to determine the status of the resources needed by an application service. The MLMs use the records of the Resource Broker for emitting events pertaining to the resources under their control to the Application Service Agent or to the management platform, respectively.

Resource Directory

The main function of the Resource Directory is to maintain a record of all resources being monitored by a particular MLM. It responds to queries and executes update operations obtained from the Resource Broker and provides persistent storage of the records. The following section describes the information flow that takes place in order to produce status views of application service offerings.

5.3 Service Management Information Flow

Figure 4 depicts the information flow once a service agent initializes itself and becomes ready to accept status updates related to the resources supporting the service monitored by that agent. The general flow is a follows:

An Application Service Agent in charge of a particular application service sends a request to the Resource Broker in order to determine the status of the resources the service depends on. It provides the Resource Broker with a list, RLa, containing the resource identifiers of all the resources that this service is dependent on. This list is generated during the application dependency analysis phase.

The Resource Broker queries the directory using RLa and obtains a list $\{(MLM_1, RLm_1),...,(MLM_i, RLm_i)\}$ of all MLMs that are responsible for monitoring the set of resources denoted by RLa. The pair (MLM_i, RLm_i) is the set of resources RLm_i monitored by the *i*-th MLM.

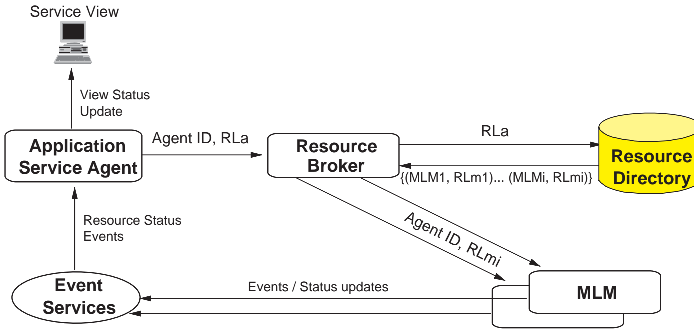

Figure 4: Information Flow between Management Components

Upon obtaining the above list, the Resource Broker contacts each of the relevant MLMs by sending the Application Service Agent ID and the associated resource list RLm_i to the *i*-th MLM. At this point, MLM *i* knows to which Application Service Agent the events related to resources in RLm_i should be forwarded.

For efficiency reasons, events from the MLMs are filtered by the event services of the management platform. Depending on the agent IDs attached to the events, the platform forwards them to the responsible Application Service Agent, which, in turn, uses the information in the event to update the status information in the service view.

6. Conclusion and Outlook

In this paper we have described an application service management framework that relies on off-the-shelf network management platforms. The work was motivated by solutions delivered for the IBM Global Network, an example of a Network Service Provider which has evolved to provide end-to-end managed application services as value-added functionality on top of basic IP connectivity services. A key requirement for application service management, highlighted in this paper, is the identification, enumeration and representation of dependency information, i.e., the dependencies that an element in the application layer has on the resources at lower layers such as middleware and network. The approach presented in this paper is pragmatic and based on a static dependency analysis that yields information on entities within a system (Intra-system) and between peer entities of a service (Inter-system). We show that standard operating systems, such as Windows NT, AIX and Linux, contain a wealth of information in their repositories that can be exploited for the *automated generation of dependencies*. Our approach has an added advantage in that it does not place any burden on the application developer to instrument their applications.

The identification of dependencies is a prerequisite for the deployment of troubleshooting services that capture fault management knowledge contained in the NSPs' fault documentation systems. Emerging technologies for building next-generation repositories such as Jini and the *Lightweight Directory Access Protocol (LDAP)* facilitate the access and dynamic exchange of management information and are currently being investigated by the authors.

References

[1] Application Management Specification. Version 2.0, Tivoli Systems, November 1997.

[2] Application Response Management. Version 2.0, Tivoli Systems, November 1997.

[3] L. Bennett. *Multiprotocol Network Management: A Practical Guide to NetView for AIX*. McGraw-Hill, 1996.

[4] P. Bhoj, S. Singhal, and S. Chutani. SLA Management in Federated Environments. In *Proceedings of the Sixth IFIP/IEEE Symposium on Integrated Network Management (IM'99), Boston, MA, USA*, pages 293–308. IEEE Publishing, May 1999.

[5] P. Brusil, J. Hellerstein, and H. Lutfiyya. Applications Management – Current Practices, Research Results, and Future Directions. *Journal of Network and Systems Management*, 6(3):361 – 366, September 1998.

[6] J. Elvers. Application Management Interface. Functional Specification, The Open Group, J.P. Morgan, Computer Associates, April 1999.

[7] I. G. Ghetie. *Network and Systems Management: Platform Analysis and Evaluation*. Kluwer Academic Publishers, 1997.

[8] B. Gruschke. Integrated Event Management: Event Correlation using Dependency Graphs. In *Proceedings of the 9th IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 98)*, Newark, DE, USA, October 1998.

[9] H.-G. Hegering, S. Abeck, and B. Neumair. *Integrated Management of Networked Systems — Concepts, Architectures and their Operational Application*. Morgan Kaufmann, 1999.

[10] IBM Corporation. *AIX Version 4.3 General Programming Concepts: Writing and Debugging Programs*, October 1997. Chapter 17: Object Data Manager (ODM).

[11] M. W. Johnson and S. Smead. *Beyond ARM 2.0 - API Extensions that enable pervasive Service Level Instrumentation*. Computer Measurement Working Group, December 1998.

[12] C. Kalbfleisch, C. Krupczak, R. Presuhn, and J. Saperia. Application Management MIB. RFC 2564, IETF, May 1999.

[13] M. Katchabaw, S. Howard, H. Lutfiyya, and A. Marshall. Making Distributed Applications Manageable through Instrumentation. *The Journal of Systems and Software*, (45), 1999.

[14] A. Knobbe, D. van der Wallen, and L. Lewis. Experiments with Data Mining in Enterprise Management. In *Proceedings of the Sixth IFIP/IEEE Symposium on Integrated Network Management (IM'99), Boston, MA, USA*, pages 353–366. IEEE Publishing, May 1999.

[15] C. Krupczak and J. Saperia. Definitions of System-Level Managed Objects for Applications. RFC 2287, IETF, February 1998.

[16] A. Schade, P. Trommler, and M. Kaiserswerth. Object Instrumentation for Distributed Applications Management. In *Proceedings of the IFIP/IEEE International Conference on Distributed Platforms (ICDP'96), Dresden, Germany*. IFIP, Chapman and Hall, 1996.

[17] S. Schiesel. Jumping Off the Bandwidth Wagon. *The New York Times*, Sunday July 11, 1999. Money and Business/Financial Desk. Section 3; Page 1.

[18] R. Sturm and W. Bumpus. *Foundations of Application Management*. J. Wiley, 1998.

[19] R. Sturm and J. Weinstock. Application MIBs: Taming the Software Beast. *Data Communications*, November 1995.

[20] D. Verma. *Supporting Service Level Agreements on IP Networks*. Macmillan Technical Publishing, 1999.