

An Architecture for Managing Application Services over Global Networks

Gautam Kar, Alexander Keller¹

IBM T.J. Watson Research Center
PO Box 704, Yorktown Heights, NY 10598

Abstract—This paper proposes a novel approach for providing end to end management of application services that have been deployed over large IP networks. A key feature of the proposed architecture is that it can be implemented as enhancements to existing, commercial network management platforms, thus allowing network service providers to offer new functionality by leveraging existing, deployed infrastructures.

Index terms—application management, network management, problem determination, application services

A. INTRODUCTION

During the last 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. In a typical scenario, depicted in Figure 1, a customer may have three locations, each with its own LAN-based IT infrastructure. These locations are labeled as Customer Premises Environments (CPE). The CPEs for a given customer are connected to the core of the service provider's global network using access point complexes, consisting of access routers, switches, etc., referred to in the figure as service provider environments (SPE), which in turn are connected to the service provider wide area network through access hubs. The entire complex is managed from one or more network operation centers (NOC).

An NSP, such as ATT, MCI Worldcom, Equant, etc., defines a virtual private network (VPN) 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 often 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. In addition, VPNs are relatively easy to install and reconfigure in case of changes. 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. 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 managed storage services, etc.

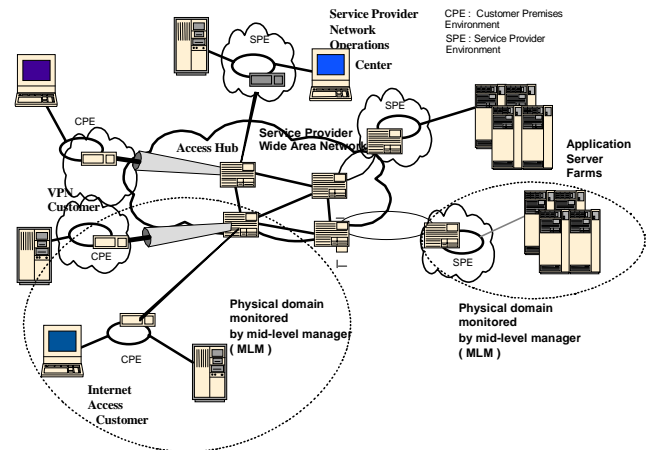


Figure 1. A Typical NSP Environment

To offer these value-added services in a cost effective manner, NSPs set up and maintain large server farms to host applications and data. The customer is offered, in addition to basic connectivity, access to "end-to-end" managed application services, associated with service level agreements (SLA) [2, 11].

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 at the application level. 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 [10]. This paper describes an architecture that addresses the above point by extending existing network management infrastructures with application service management capabilities, i.e. an architecture that exploits network management functions to provide fault and performance management for networked application services.

Section B gives an overview of a typical service provider network and its management architecture. Section C introduces the notion of application services and defines the concept of *service management domains*. Within these service management domains, dependencies exist between different kinds of resources [8, 9]. The generated dependencies are used as input for our application service management architecture whose elements are described in section D. Section E refers to related work and Section F concludes the paper and presents issues for further research.

¹{gkar|alexk}@us.ibm.com

B. BASIC NETWORK MANAGEMENT FOR NSPs

The general approach to managing a large global network is to partition the physical network environment into what is known as *monitored domains*, each of which is under the supervision of a (typically SNMP-based) management system, often called mid level manager (MLM). These monitored domains are typically defined according to topological considerations. Figure 2 illustrates the concept using an example where two server farms are being used to provide various application services to different customers.

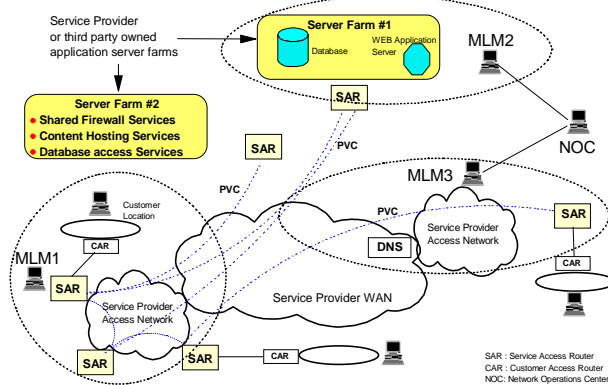


Figure 2. NSP environment and its management domains

Here the physical network has been partitioned into three monitored domains, each under the supervision of an MLM. 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 monitored domains. Therefore, monitoring the health and status of an application service requires coordinated monitoring and collection of selected information across multiple physical domains. In other words, in order to effectively manage networked application services, there needs to be a system that will be able to correlate monitored data at the resource level with the performance of the managed application services. 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. An example of a hierarchical management structure that is commonly used to address this problem is shown in Figure 3. Standard network management platforms, such as Tivoli TME, HP OpenView and Cabletron Spectrum provide for such a hierarchical, distributed architecture [1, 4, 7].

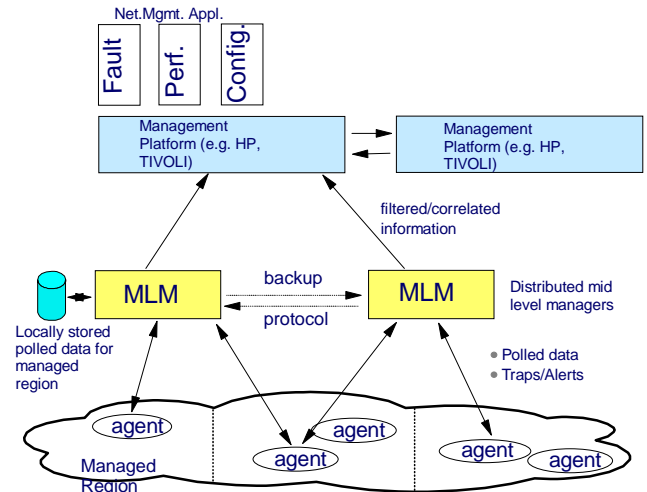


Figure 3. Hierarchical network management for NSPs

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 viewpoint, they appear as extended SNMP agents to the top-level manager at the Network Operations Center (NOC).

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. In our scenario, this policy is refreshed once a day to reflect changes in the 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.
- correlation of events for the domains under its control. There are a number of different correlation algorithms that can be used by the MLMs, depending on need and applicability. This paper will not discuss this function any further. The reader is referred to the literature where there are a number of papers available on the subject. See e.g., [6].

- 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. Basically, the MLMs have been grouped into pairs, where the members back each other up. They exchange heartbeat messages at periodic intervals and if one does not hear from the other, it takes over the monitoring responsibility for resources in the failed MLM's domain. This exchange happens through the services of a resource directory, as explained in section D.

Note that the above are typical functions performed by an MLM. Not all these functions are necessary or relevant to the algorithms discussed in this paper.

Today, many 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 analyze in the next section, this transition brings forth important requirements.

C. APPLICATION SERVICE MANAGEMENT

1) *Application Service Management Requirements*

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 [2, 11], 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, so that the following key functions may be provided:

Root cause analysis for application service problems: Depict a particular customer's service status, much the same way as one is used to seeing the status of a network resource. This component 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 for application services: When a resource goes down, determine as quickly as possible 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 system (CRM) to issue proactive warnings to affected customers.

The next section introduces the concept of *service dependencies* that help to address the above issues and form an integral part of the application management architecture that is being proposed in this paper.

2) *Dependency Analysis for Service Management*

An important requirement in being able to perform effective fault and performance management in a networked environment is the ability to understand how faults occurring in the resources of one layer affect the working of resources in another layer. In particular, for *application service management*, it is crucial that one be able to capture and record the dependencies a particular application service has on the underlying network resources. These requirements are best illustrated by the example shown in Figure 2, 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 find it very difficult to deal with service dependencies because topological domains for network management differ greatly from service management domains with respect to 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 not scalable. It is error-prone and tends to yield inconsistent data. As motivated in sections A and B, 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 and, possibly, inconsistency.

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 establish 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., [6]). Our approach is based on a static dependency analysis technique that yields a graphical representation of how an application service is dependent on lower layer services/resources in a networking environment. For example, Figure 4 depicts a typical dependency graph for a three tier e-commerce environment. The entity at the tail of an arrow is dependent on the services provided by the entity at the head of that arrow, e.g., the web application service function depends on the web service and the database service.

For the rest of the paper we will assume that this dependency information has been obtained, processed and represented in a dependency database [8]. The service management architecture presented in this paper will make use of this information through the concept of service management domains explained in the next section.

3) Service Management Domains

Resources contributing to the functioning of an application service belong to different physical domains, each under the monitoring scope of a MLM. In Figure 2, MLM1, MLM2 and MLM3 monitor the resources that need to be operable for the content hosting service to be functional.

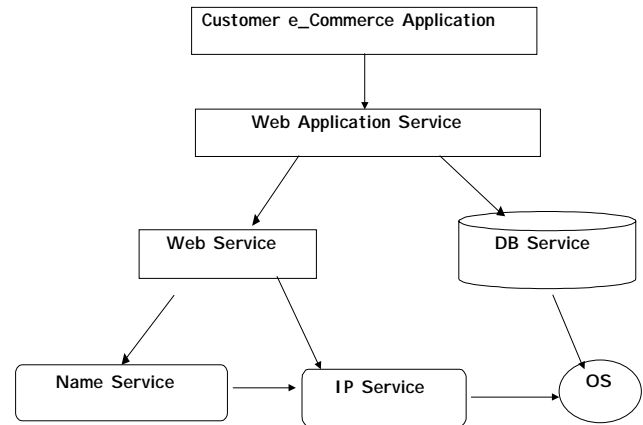


Figure 4. A Typical Dependency Graph

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 D, our architecture uses the concept of a *resource broker* in conjunction with a *resource directory* to address this issue.

As depicted in Figure 2, each MLM is responsible for monitoring a set of resources. Application services depend on the functions provided by lower layer services and, therefore, on the proper operation of lower layer resources. Thus the resources that cooperate to provide the underlying functions required for delivering an end to end application service, belong to the domains of multiple MLMs. The collection of resources from different physical MLM domains that affect the functioning of an application service is called a *virtual service management domain*. For example, an end to end service, such as content hosting, provided by server farm #1 (as depicted in Figure 2) is dependent on (in brief):

- the servers in the server farm and the SAR that connects the farm to the service provider WAN, under the domain of MLM2,
- the DNS service and its associated resources under the domain of MLM3
- the resources in the service provider access network to which the customer (as shown in the figure) is connected, under the domain of MLM1.

Thus, by our definition, the virtual service management domain for the content hosting service, as shown in Figure 2, will consist of all the above resources, spanning the control of three different MLMs.

Figure 5 depicts the relationship between the physical monitored domains in the network and the virtual management domains defined in the Service Management layer of our architecture. As shown in the figure, the main entity that is responsible for managing a service is the Service Management Agent which monitors its associated virtual service management domain, with the help of information that it receives from the associated MLMs. Section D, which

describes the architecture in detail will show how this is achieved.

Once we have the list of resources that a particular service depends on, we need to discover the MLMs which are responsible for monitoring those resources. In other words, each service management agent is logically connected with a set of MLMs, those that are responsible for monitoring the resources which cooperate to deliver the service. These MLMs can then report on the status of these resources to the relevant Service Management Agent, which updates its view of the service.

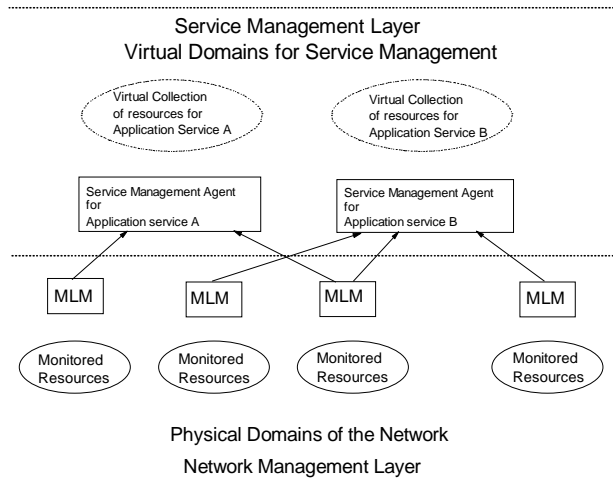


Figure 5. Service Management Domains

At this stage, from an architectural design viewpoint, the following important questions arise:

- which mid level managers (MLM) report status to a given service management agent?
- how does a service management agent build its virtual domain of managed resources?

The next section addresses these questions within the framework of our proposed application management architecture.

D. APPLICATION MANAGEMENT ARCHITECTURE

1) Dependency Information

As a result of the static analysis [8, 9] performed on installation repositories and other sources of information in a deployed application environment, a dependency graph is constructed and stored in a dependency database. By querying this dependency database, an application service agent is able to construct a list of all resources that the application is dependent on. The end result is that each application service offering has associated with it a list of resources in the network management layer that provide the base for that service. This data is kept in the application analysis database as shown in Figure 6.

2) Components of the Architecture

Figure 6 represents the essential components of the application service management architecture. As discussed at the beginning of this paper, the point to note is that this architecture is built on top of standard network management platforms (HP OpenView, Tivoli NetView, etc.) and is therefore reasonably inexpensively implementable by service providers, by leveraging the functions of their existing network management systems.

(a) Application Service Agents

The application service agent is the focal point for managing an individual service offering. If, for instance, the NSP has three different service offerings, our design yields three service management agents, each responsible for one offering. From an implementation viewpoint, these agents are like applications operating on top of a network management platform, that is, they can use the basic services of a network management platform, such as event services, topology services, etc., just as typical user developed network fault or performance management applications do today.

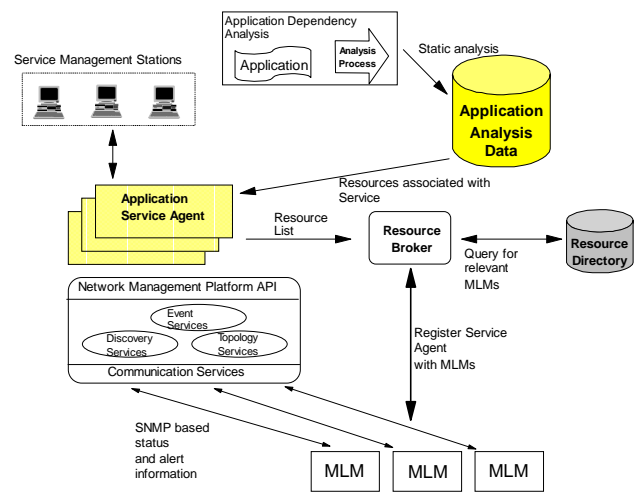


Figure 6. Application Service Management Architecture

The functions of an application service agent are:

- When a particular application service agent initializes itself, it queries the dependency data base to construct a list of resources that this application depends on. For instance, in the case of the content hosting scenario discussed in section B (and shown in Figure 2), the resource list associated with this application contains all the resources that support the IP connectivity, the back-end database, the NFS and the DNS.
- 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 graph, 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 service management agent also maintains a list of customers who have currently subscribed to a given service, so that it can correlate customer complaints with service outages.

(b) *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 information received from the Resource Broker for delivering events pertaining to the resources under their control to the Application Service Agent through the management platform.

(c) *Resource Directory*

The main function of the Resource Directory is to maintain a record of all resources being monitored by a particular MLM. In order to respond to queries, as described in the next section, it maintains two views of the monitored resources. One is by the MLM identification and the other is by resource identification. Thus two typical record types in the resource directory are:

{**MLMi**: $R_{i1}, R_{i2}, \dots, R_{in}$ }, which indicates that MLM_i monitors resources R_{i1}, R_{i2}, \dots , etc.

{**R_i**: MLM_{i1}, MLM_{i2}, ..., MLM_{in}}, which indicates that the resource R_i is monitored by MLM_{i1}, MLM_{i2}, etc.

The resource directory 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. In general, the resource directory can keep track of which services are being offered by which provider, so that the system can be used in a multi provider environment.

3) *Service Management Information Flow*

Figure 7 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 figure shows some of the essential interactions between the key building blocks of the application management architecture. The main portion of this enhancement is the Application Service Agent, which, viewed from the network management platform, is just another standard application. In Figure 7, as far as the main flow of information is concerned, the only component of the network management platform that is involved in this interaction is the event services. This component, as in typical network management platforms, is able to filter events based on set criteria and is able to forward events to specific applications, in this case the Application Service Agents, based on their identifiers.

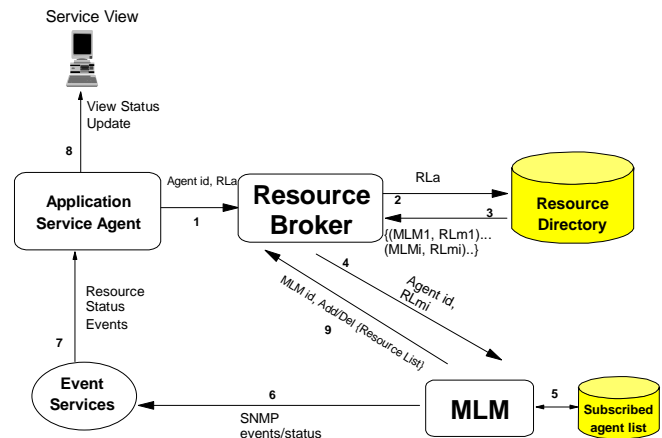


Figure 7. Application Management Information Flow

The general flow is as follows (corresponding to the numbered arrows in Figure 7) :

1. 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 its identifier and a list, RL_a, containing the resource identifiers of all the resources that this service is dependent on. This list is generated during the deployment phase when the agent consults the dependency database (see the part on application service agents towards the beginning of this section).
2. The Resource Broker queries the Resource Directory using each element present in RL_a asking for the ids of the MLM responsible for monitoring these resources.
3. The Resource Directory responds with a List, {(MLM₁, RL_{m1}), ..., (MLM_i, RL_{mi})}, of all MLMs that are responsible for monitoring the set of resources contained in the set RL_a. The pair (MLM_i, RL_{mi}) denotes that the set of resources, RL_{mi}, is monitored by the i-th MLM.
4. At this point the resource broker has a list of MLMs each of which monitors a subset of the total set of resources that the particular application service is

dependent on. The Resource Broker contacts each of the relevant MLMs by sending the Application Service Agent ID and the associated resource list RL_{mi} to the *i*-th MLM.

5. At this point, MLM_{*i*} knows to which Application Service Agent the events related to resources in RL_{mi} should be forwarded. It records this information in the subscribed agent list database, shown in Figure 7.
6. When an event related to a particular resource is generated, the MLM responsible for monitoring that resource is notified of the event (note: in some cases, multiple MLMs can be responsible for monitoring the same resource; this does not affect the information flow). This MLM consults the subscribed agent data base and obtains a list of service management agents that are interested in obtaining status information about the involved resource. It attaches the IDs of these service management agents to the event and forwards the information to the network management platform (the event services component of the platform receives this information).
7. Depending on the agent IDs attached to the events, the event services component of the network management platform forwards them to the responsible Application Service Agents. This is possible because these agents have registered themselves with the event services component as subscribers. Note that for efficiency reasons, events from the MLMs may be filtered by the event services of the management platform, depending on filter characteristics set by the Application Service agents. Most management platforms allow the setting up of such filters.
8. Upon the reception of an event related to a resource which is a contributor to the functioning of its monitored service, the Application Service Agent uses the API of the service view station to update the status view of the application. A human operator or a programmed management function can use this service view to perform corrective management actions (this part is not discussed in the paper and is only mentioned for completeness).
9. This particular flow (marked 9 in the figure) is separate from the main flow in that this occurs between an MLM and the Resource Broker and is unrelated to the occurrence of an event. It is quite common in a networked environment to add or remove resources. For example, when a new customer is brought on board, new links and access routers are deployed. Additionally, when traffic engineering recommends, new content hosting servers may be added to the server farm. Thus it is common to add or delete resources to the monitoring list of a particular MLM. When this happens, the MLM in question informs the Resource Broker about the change, so that the list of monitored resources associated with this MLM can be updated in the Resource Directory. The reader should note that this

list needs to be kept current since it is used in steps 2 and 3 in the general information flow. Additionally, a similar flow (not shown in the figure) between the MLMs and the Resource Broker is used to support the backup protocol between MLM pairs (see reference to this aspect in section B) where an MLM that takes over for a failed partner obtains the monitored resource list of the partner from the Resource Broker.

E. RELATED WORK

The OpenGroup *Distributed Software Administration (XDSA)* specification [12] addresses the mechanisms for software distribution/deployment/installation by defining several commands (swinstall, swlist, swmodify etc.) and a software definition file format with many attributes. The latter identifies three kinds of relationships: prerequisite, exerequisite, corequisite. It should be noted that XDSA does not deal with *instantiated* applications and services and therefore does not represent means of determining the dependencies between components at runtime.

In the *Common Information Model (CIM)* [13], dependencies (being usually perceived as a specific kind of association) are modeled as classes, thus allowing inheritance. The following dependencies are specified in the different CIM schemas:

- The *Core Schema* defines dependency types in terms of abstract classes that deal with dependencies between SAPs, services, products and provide a generic means for associating context with them. The *TypeOfDependency* attribute describes for a given service that its antecedent “must have completed”, “must be started” or “must not be started” in order for a service to function. This is related to the notion of prerequisites, corequisites and exerequisites in XDSA.
- The *System Schema* refines the root class CIM_Dependency in order to deal with job destinations, host systems and file systems.
- The *Application Schema* defines two dependency types that describe an association between a service and an SAP and their implementations, respectively. Finally, the *Distributed Application Performance (DAP) Schema* relates the definition, the metrics and the logical element that is instantiated to a “unit of work”.

All schemas have in common that their dependencies are derived from a single base class CIM_Dependency being defined in the Core Schema; the inheritance hierarchy is therefore flat.

It is fair to say that while each specification addresses the dependency problem in general, none of these specifications allows the determination of dependencies *at runtime*: In XDSA and CIM, the dependencies are well specified for the installation phase of a software product. However, these models provide no support as soon as an application gets instantiated, i.e., moves from the “installed” state to the

“running” state. The main reason for the absence of satisfactory application management solutions is that comprehensive application management demands a large amount of management information, thus posing an additional development effort on the application developers [14]. A good example for this is the *Application Management Specification (AMS)* [15] 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.

The authors are engaged in applying the modeling methodology of CIM to the enumeration and representation of the types of dependencies addressed in this paper.

F. CONCLUSION

In this paper we have looked at the evolving application service provider environment and have proposed an architecture for managing networked application services. While this architecture can be used for fault, performance and configuration management, through the use of dependency information, we have concentrated on fault management and on the recognition of events emitted by resources that can impact a service. A key feature of our architecture is that it can be implemented using standard network management platforms. Some of the problems addressed in this work are based on experience with the IBM Global Network (currently a part of AT&T Solutions), 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. Such NSPs can use our framework to offer management functions for application services by enhancing their existing network management platforms and infrastructures, thus reducing the cost of new deployment by leveraging existing functionality. A key requirement for application service management, mentioned 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. In order for our architecture to work, such dependency information has to be gathered and represented in a query-able data base. For details on this process, the reader is referred to [9].

An important feature of a managed, networked application service is that the same infrastructure components (networks, servers, application components, services) are shared among multiple customers, each with different Service Level Agreement (SLA) requirements. Our proposed architecture, through the functionality of the Service Management Agent, is able to provide customized service status views for individual customers. Future work in this area needs to investigate how such a setup can be enhanced to include the computation of customized SLA statistics. Questions such as, “what effect is there on current SLAs when new customers are added to an

existing service?”, “what are some critical scalability issues?”, etc., need further investigation.

One of the concepts presented in the paper is that of a *virtual service management domain* which consists of resources from different physical domains, monitored by individual stations, called mid level managers. A physical resource can, in general, be a part of multiple virtual domains, depending on how the services are defined and deployed. The monitored information for a physical resource is distributed to the relevant service management agents. One interesting area of investigation would be to understand how the different service management agents can provide feedback to the monitoring engines (in our case the mid level managers), so that monitoring policies can be changed depending on service performance or status. These and other related areas are being investigated by the authors.

G. REFERENCES

- [1] L. Bennett *Multiprotocol Network Management: A Practical Guide to NetView for AIX*. McGraw Hill, 1996
- [2] 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
- [3] 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.
- [4] I. G. Ghetie. *Network and Systems Management: Platform Analysis and Evaluation*. Kluwer Academic Publishers, 1997.
- [5] R. Gopal. Layered Model for Supporting Fault Isolation and Recovery. In *Proceedings of the 2000 IEEE/IFIP Network Operations and Management Symposium (NOMS2000)*, Honolulu, HI, USA, pages 729 – 742, April 2000.
- [6] 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.
- [7] H.-G. Hegering, S. Abeck and B. Neumair. *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann, 1999.
- [8] G. Kar, A. Keller and S. Calo. Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis. In *Proceedings of the 2000 IEEE/IFIP Network Operations and Management Symposium (NOMS2000)*, Honolulu, HI, USA, pages 61 – 74, April 2000.
- [9] A. Keller and G. Kar. Classification and Computation of Dependencies for Distributed Management, The Fifth IEEE Symposium on Computers and Communications (ISCC 2000), July 4 – 6, 2000, Antibes, France.
- [10] M. Katchabaw, S. Howard, H. Lutfiyya and A. Marshall. Making Distributed Applications Manageable through Instrumentation. *The Journal of Systems and Software*, (45), 1999.
- [11] D. Verma. *Supporting Service Level Agreements on IP Networks*. Macmillan Technical Publishing, 1999.
- [12] Systems Management: Distributed Software Administration. CAE Specification C701, The Open Group, January 1998.
- [13] Common Information Model (CIM) Version 2.2. Specification, Distributed Management Task Force, June 1999.
- [14] M. Katchabaw, S. Howard, H. Lutfiyya, and A. Marshall. Making Distributed Applications Manageable through Instrumentation, *The Journal of Systems and Software*, (45), 1999.
- [15] Application Management Specification. Version 2.0, Tivoli Systems, November 1997.