SLA Management in Federated Environments

Preeti Bhoj, Sharad Singhal, Sailesh Chutani
Internet Systems and Applications Laboratory
HPL-98-203
December, 1998

federated management, service level agreements, system monitoring

Increasingly, services such as E-commerce, web hosting, application hosting, etc., are being deployed over an infrastructure that spans multiple control domains. These end-to-end services require cooperation and internetworking between multiple organizations, systems and entities. Currently, there are no standard mechanisms to share selective management information between the various service providers or between service providers and their customers. Such mechanisms are necessary for end-to-end service management and diagnosis as well as for ensuring the service level obligations between a service provider and its customers or partners.

In this paper we describe an architecture that uses contracts based on service level agreements (SLAs) to share selective management information across administrative boundaries. We also describe the design of a prototype implementation of this architecture that has been used by us for automatically measuring, monitoring, and verifying service level agreements for Internet services.

1 Introduction

Increasingly, services such as E-commerce, web hosting, application hosting, etc., are being deployed over an infrastructure that spans multiple control domains. These end-to-end services require cooperation and internetworking between multiple organizations, systems and entities. As shown in Figure 1, even within a single enterprise, multiple organizations (or geographically distributed sites) can maintain independent management systems that need to share management information. Cross-domain management is especially critical when outsourcing Information Technology (IT) services or when extending business applications to systems in other enterprises in order to form extended enterprises.

Managing large-scale applications that cross such administrative boundaries is a problem because current management solutions either allow partners access to all management information (e.g., by providing remote consoles) or deny access to this information. Inter-domain management in competitive environments places two fundamental requirements on the management system:

1. Service management and diagnosis require the knowledge and view of the end-to-end service. This means that management information has to flow across administrative domain boundaries to provide an end-to-end view.
2. Business requirements restrict information sharing across domains because the details of the service implementation and much of the customer information is considered proprietary by each business. Thus business policies restrict the sharing of details about components and infrastructure used in delivering the service. These policies are particularly stringent if customer data needs to be shared across domains.

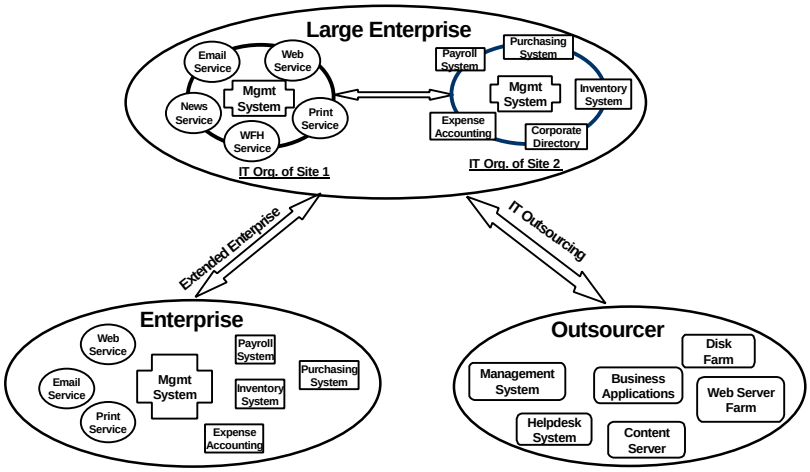

Figure 1. Internet Business Environment

Meeting these objectives requires that service management systems need to

- Selectively share information about the components of the overall service only to the extent necessary to ensure overall operation of the service.
- Hide the details of the system components by abstracting information.
- Provide mechanisms to ensure that service level obligations provided by the administrative domain to its customers and partners are being met.

Over the years, a number of solutions have been proposed for inter-domain management. The telecom industry and the ATM/POS network providers have coordinated and shared information between multiple entities. Unlike the Internet their environments are regulated and are typically designed to offer a single service. Differences between these services and Internet-based services are discussed in [1].

Recently, work has also been done on inter-domain information sharing on the Internet. In [2], a trouble-shooting methodology for coordinating network problem diagnosis among peer administrative domains and

untrusted observers is presented. Mechanisms to manage security in a heterogeneous multi-provider environment are discussed in [3].

Inter-domain communication is discussed in TMN. There are a number of publications in the area of using the TMN model to manage Virtual Private Networks (VPN). The design of a management service for a VPN that addresses multiple domains and heterogeneous systems are discussed in [4]. In [5], approaches in Internet services management are compared to those taken by the telecommunication industry (TMN). It is pointed out that more effort is required to achieve a standard Internet service management strategy to manage all types of internet services as well as non-internet based services such as voice services.

When describing service level management, one of the most commonly used service metric is availability. [6] describes methods for testing the availability of distributed applications by constructing a service graph for the description of functional dependencies and applying calculation rules on an instantiated graph to determine the availability of applications.

Most published work in this area describes approaches to managing and trouble-shooting network services, and managing security, in an inter-domain environment. A few products (e.g., InfoVista systems [7], Netcool [8], Vital Analysis[9], Network Health Reporter[10], etc.) allow customers to monitor the quality of service offered by providers.

While the research and products mentioned above offer a good start towards Internet service management, there still remain unresolved problems:

- None of the research addresses how to selectively share management information across administrative domain boundaries in a secure way. This capability is particularly important with the introduction of extended enterprises where a service is composed of components from several service providers.
- There are no tools available to derive measurable aspects from Service Level Agreements (SLAs). It is unclear how a legal service level agreement document is translated into a measurable specification that can be automatically monitored for compliance.
- There are no recommendations and policies to define metrics (what they are and how their values are computed) and their bounds (thresholds, baselines, etc.) for service compliance.

Work is underway in the IPPM (Internet Protocol Performance Metrics) working group of the IETF, the XIWT (Cross Industry Working Team) and ANSI T1A1 committee to identify Internet service related metrics, and measurement methodologies. We have focussed our work on developing mechanisms to share selective management information across federated domain boundaries, and measuring, monitoring, verifying, and managing service level agreements for Internet services. We expect our solution to complement and work side by side with traditional management solutions, such as HP OpenView, CA Unicenter, and Tivoli TME. We assume that the measurements collected by the management and measurement systems can be combined to service level metrics used in the service level agreements.

In section 2, we discuss our overall architecture for sharing information in federated systems. Section 3 describes contracts, which are used in our architecture for encapsulating measurable aspects of service level agreements for the purposes of management. The design of *Conformance*, a prototype system to monitor SLAs for compliance and provide selective sharing of management information is discussed in section 4. Finally, we describe an example where we have used Conformance to monitor Internet services in section 5 and finish with conclusions in section 6.

2 Federation Architecture

A *federated system* is defined to be a system composed of components within different administrative entities cooperating to provide a service. A *service* is an application with a well-defined interface and functionality. *Federated service management* is the management of services that span multiple heterogeneous control domains, and which rely on correct functioning of components across those domains. A *control domain* is defined to be an administrative domain that is managed by a single administrative entity, typically a business. [1] elaborates on these concepts in greater detail.

Service providers are increasingly using SLAs to define agreements for sharing resources with partners, as well as for offering service quality guarantees to customers. These SLAs contain (along with other legal obligations) details of information that is shared and service level guarantees that are offered by the

provider. Management systems should contain information about these SLAs, and should use this information both to control access to system resources as well as to monitor the system for compliance with the SLA. Ultimately, the management system should control resources to actively manage the services with the objective of meeting these agreements.

We have developed an architecture to allow SLA monitoring and sharing of selective management information across administrative domain boundaries in a secure way. In our architecture we assume that all interactions between federated domains are based on bilateral agreements that can be implemented using verifiable and consistent contracts. A *contract* is a specification (derived from the SLA) of the service attributes that are meaningful and automatically measurable for correct service behavior. The contract specification contains both the attributes and the bounds within which the attributes must stay in order for the service to behave in a desired manner. Attributes have to be both quantifiable and measurable to be included in a contract. Different domains are likely to contain heterogeneous systems, it is thus important that there be agreement on how contracts will be invoked and how the systems will communicate. We describe contracts in further detail in section 3.

Figure 2 shows a high level overview of our architecture. In this section we explain the diagram with general descriptions of each of the components.

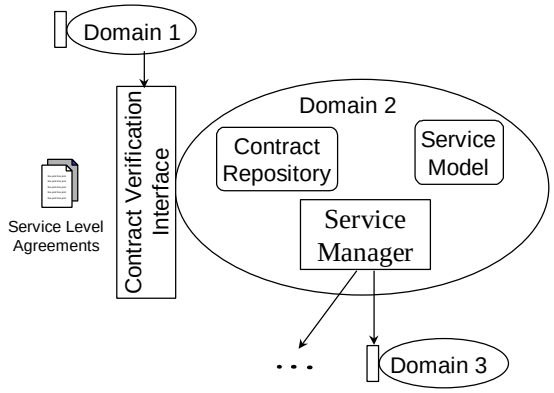

Figure 2. Federation Architecture

The ovals represent administrative domains. All external interactions to a given domain happen through one or more Contract Verification Interfaces provided by the domain. Service level agreements offered by the domain define the nature of information provided at the interface and the security parameters needed for the interaction. The smaller oval at the bottom represents another service domain, which might either be self-contained within a single business entity, or might be federated. The architecture provides a recursive and hierarchical model for communication in a federation, thus providing a scalable solution. It is assumed in the architecture that each domain controls only the service aspects that are provided by it.

Inside the domain, the service manager directs the collection of management information using the service-specific data contained in the service model. The data collection is guided by the contracts in the contract repository.

The *service model* includes a description of the service that this administration is managing. It

- Identifies the various components that enable a service. For example, if the top level service being managed is electronic mail, the service model would list the components as the email server host, the networks connecting the email host to the internet, the email application itself, the name server used to resolve hostnames to internet addresses, etc.
- Expresses interdependencies that exist among the different elements of the service. From the above email example, all components identified in the service model should function properly for the email service to work. The interdependencies capture the cause and effect between the components of a service.

- Identifies the measurements that are available from each component. Thus, the email server could identify the number of email transactions, and active measurements could be used to get an estimate of the response time seen by email clients.

The *contract repository* contains a set of contracts, which this domain has with its providers and customers. It contains information on how to validate an incoming contract verification request and places constraints on what data may be accessed from outside the domain as well as how that data is computed. As mentioned earlier, contracts consist of a specification of attributes and bounds within which the attributes must stay in order for the service to behave correctly.

The *service manager* is the engine responsible for directing the verification task. It has the knowledge of how to evaluate an incoming contract verification request. This would involve interfacing with the contract repository to get the details of the contract, and interfacing with the local resources and local system and service management modules to collect information needed to verify the contract. If evaluation of a contract has dependencies on other external contracts then it uses the contract verification interfaces provided by those external domains to collect the data.

Figure 3 shows how the local resources and management systems are coupled to the contract verification interface to expose selected information. The local measurement and management systems collect data from the local infrastructure and applications. Customizable plug-ins allow the service manager to communicate with a variety of systems to extract information about the domain. The plug-ins provide an abstract view of the system to the service manager that is independent of the underlying implementation. The contract evaluation and notification handlers use this abstract view to compare the system behavior to pre-set thresholds and conditions specified in the contracts to monitor the contracts for compliance.

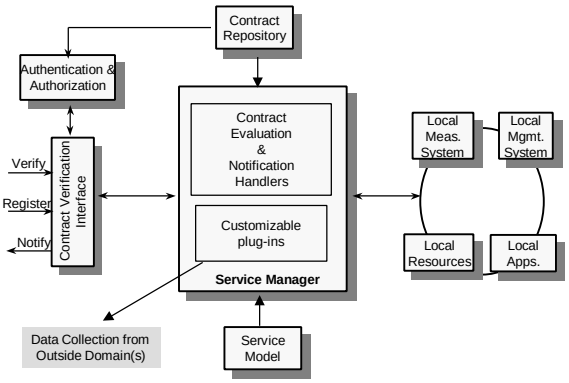

Figure 3. Contract Verification Framework

3 Contracts

Contracts govern the details of which data can be exposed and to whom. In this section, we describe the functionality offered by contracts in more detail.

3.1 Contract Definition

Formally, a contract *C* is defined by the triple (*P, M, A*), where *P* is the set of *properties* associated with the contract, *A* is the set of *assertions* agreed upon by the parties, and *M* is the set of *methods* (or operations) available on the contract.

Properties define information (needed for contract verification) that does not necessarily relate directly to the specific service that is the subject of the contract. Examples of items in *P* are

P = {*authentication mechanism, access control, invocation methodology, …* }

Assertions contain service-related agreements or guarantees. The assertion set *A* consists of

$A = \{\mathcal{R}(\boldsymbol{v})\}$

where $\boldsymbol{v}$ is a vector of variables that reflect some aspect of the service and $\mathcal{R}(\boldsymbol{v})$ is a relationship that constrains those variables according to contract agreements. At any given time, $\mathcal{R}(\boldsymbol{v})$ takes on the values TRUE or FALSE depending on whether the constraint described in the relationship is being met or not. Examples of assertions in *A* may be

$A = \{availability > 99.9\%, packet\ loss < 2\% \wedge round\ trip\ delay < 150\ ms, \ldots\}$

Methods describe the operations available on the contract at the contract verification interface. They permit the invoker to query the truthfulness of the assertions in the contract, identify assertions that are FALSE, and retrieve the values of associated variables in the assertions. For example, *M* may consist of

$M = \{verify\ contract, query\ variable\ values, register\ constraint, notify\ event, \ldots\}$

Contracts are described by a *Contract Definition Language* (CDL) which gives a formal declaration of the assertions in the contract, and allows the service manager to associate the contract with a number of *handlers*, which hide the details of the service and the means of verification. Contract structure and verification is discussed further in sections 4 and 5 using examples.

3.2 Contract Content

From an operational point of view the set of *assertions* in the contract defines its content. Each assertion is an atomic group of statements that is agreed upon by the parties defining the contract. At any given time, an assertion may be TRUE or FALSE depending on whether the party is meeting the obligations stated in the assertion or not. Statements in an assertion are made up of *logical predicates* whose value can be uniquely determined. The logical predicates are composed using *variables* as well as *logical operators* such as $\{\wedge, \vee, \neg\}$, *quantifiers* $\{\forall, \exists\}$, *set operations* $\{\in, \cup, \cap\}$ and *constraints* such as $\{\leq, \geq, \neq, =\}$ on those variables. Variables may be simple variables (e.g., current network load), statistical variables (e.g., averages or variances), or trends (time dependent variables such as growth rates). They reflect measures that are meaningful for the operation of the service.

A contract is said to be *in compliance* if all assertions within it are TRUE. This requires that all assertions in the contract should be *verifiable* and *consistent*. We call an assertion verifiable if means exist to programmatically compute whether it is true or not at any given time. For a set of assertions to be consistent, we require that no dependencies exist within the set such that compliance of one assertion forces non-compliance in another.

3.3 Contract Verification Interface

Contracts are verified using a *Contract Verification Interface* that describes the set of operations that may be invoked across the domain boundary during system operation. Since contracts govern both the behavior of the interaction between domains as well as the nature of information that is exchanged between domains, the verification interface can potentially be very complex. However, the problem of specifying the interface becomes simpler if we note that not all information in the contract is dynamic. Most of the contract content mentioned in the earlier section is agreed upon ahead of time, and remains static (and known to both domains) during operation. Typically this information includes:

- What are the service quality metrics and service guarantees and what thresholds will be met?
- What specific information will be shared across domains, i.e., what parameters will be passed and what results will be returned?
- What are the methods for communications between systems and what authentication and access control methods are acceptable?
- What are the arbitration policies and what information will be checked for audits?

Thus, this information does not need to be communicated across the domain boundary at run-time but can be defined using an administration interface. A description of the requirements and desired features for such administrative tools is beyond the scope of this paper.

At a high level, we believe that only two capabilities need to be supported at the contract verification interface: *contract verification* and *event notification*. The interface is described in terms of an API

consisting of parameters and operations that are invoked across the domain boundary during the operation of the system.

4 Conformance Prototype

In the previous sections we described an architecture for sharing information in federated environments based on service level agreements. We also defined a contract-based mechanism to specify the information needed to monitor the SLAs for compliance. Contract interfaces can also be used for sharing other information between administrative domains. The architecture leaves open the exact communication and security mechanisms to be used, as well as methods for specifying contracts and coupling them to local management systems.

In this section, we describe *Conformance*, a prototype implementation of this architecture. Conformance is specifically targeted to allow automatic verification of quality of service guarantees as described in a contract. The implementation of Conformance is web-based, i.e., inter-domain communication uses the HTTP protocol. This allows the use of existing facilities provided by web servers and clients for data encryption using secure socket layers (SSL) and authentication using public key certificates. Conformance is written in Java, and uses a Java-enabled web server as its front-end. The verification requests are authenticated at the web server, and the request parameters are passed back to Conformance through a Java API. Results from Conformance can be passed directly back to the client, or in the form of HTML reports. We have tested our implementation on both NT and Unix platforms.

4.1 Conformance Process Flow

Figure 4 shows the process used to create the data necessary for contract verification using Conformance.

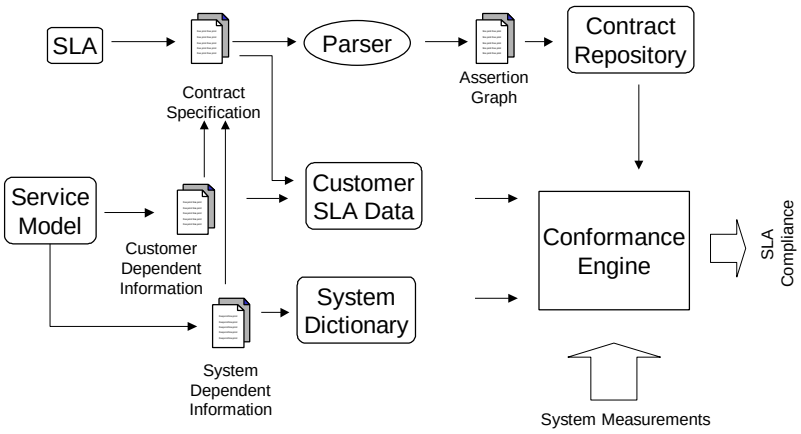

Figure 4. Conformance Process Flow

The service model describes the service implemented by the domain as well as the dependencies between service components. The System Dictionary, which is a part of the service manager, contains the abstract view of the service model in terms of high-level service attributes that are offered to customers. The System Dictionary is used to identify the attributes as well as meta-information about the attributes such as which measurement plug-in is used to obtain the attribute value, how often the measurement is made, what parameters are necessary to measure it and so on. Thus, for example, for a *packet loss* measurement, the attributes could specify that it has a TTL (Time-To-Live) of 15 minutes, that it requires a network segment identification as a parameter, and that it should be obtained using the *Network_Measures* plug-in. This way the System Dictionary isolates the Conformance Engine from the details of the underlying system implementation.

For each customer SLA, information about the guarantees (thresholds and bounds on the attributes) as well as information about the service components which impact the service offered to a given customer are

placed in the Customer SLA Database. An example of the former could be *guaranteed availability = 99.9%* while an example of the latter would be *premises router address = 15.25.0.0*.

A Contract Definition Language (see section 4.3) is used to specify the assertions in the form of a template that uses the attributes and thresholds as parameters. The contract is compiled into a graph structure that is then added to the Contract Repository.

When a contract is to be verified, the contract template is retrieved from the contract repository, and the customer specific information is retrieved from the customer SLA database. This information is then passed to Conformance. Conformance uses the information in the System Dictionary to make (or retrieve) the appropriate system measures and computes the compliance with the contract.

4.2 Design of Conformance

The overall structure of the Conformance Engine is shown in Figure 5.

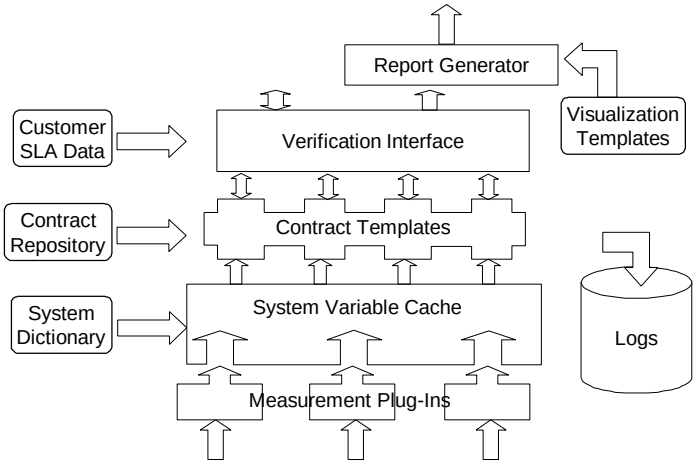

Figure 5. Design of Conformance

Objects that contain contract templates form the core of the engine. Because the same contract could be offered to multiple customers, customer-specific thresholds, bounds, and system parameters are filled in the template at contract evaluation time. This allows the templates to be shared across multiple customers.

System measurements are accessed through system specific measurement plug-ins. Because the plug-ins hide the details of the underlying system and the measurement protocols from Conformance, new measurements are easy to add to the system. Conformance dynamically loads measurement plug-ins as needed.

To minimize measurement traffic, the state of the system is cached in the system variable cache. The cache uses a logical database view of the system, i.e., every system attribute is treated as an entry in a logical database, with the attribute name being used as the key to retrieve its current value. Properties associated with the attribute define how frequently the value is updated in the cache, which parameters are needed to obtain the measurement value, and which measurement plug-in is responsible for obtaining the attribute value. The cache can also store measurement history if needed to compute aggregate values (e.g., time averages) if they are not directly available from the underlying measurement system.

When a customer request is received at the verification interface, the following steps occur:

1. The customer specific data is retrieved from the customer SLA database and inserted in the contract templates.

2. The actual system attribute[1] values are obtained from the system variable cache. If the cached values are stale, measurement plug-ins are used to update the cache before the values are returned.
3. The contract is evaluated by computing the values of the assertions defined in the contract using the attribute values and customer-specific parameters, thresholds and bounds.
4. The compliance results are logged and communicated either programmatically by the verification interface, or through reports generated by the report generator. The reports are typically customized for each customer. The report generator uses the Visualization Templates to create reports.

4.3 Contract Definition Language

Systems can be defined using a formal language such as UML [11]. Other system modeling languages used in CIM [12] and Webm [13] can also be used to describe and collect management data using multiple heterogeneous sources of data such as SNMP, CMIP, DMI, etc. In our implementation, we assume that these data collection mechanisms can be captured in one or more measurement plug-ins and the system attributes can be derived from those measurements.

Since we did not need the complexity enabled by UML, we used a declarative language with syntax similar to the C language. A partial description of the grammar used is shown below:

```
Contract
      : DeclarationList AssertionList [ FilterList ]
Declaration
      : ContractName | ServiceName | Variable Type Declarations
Assertion
      : PredicateList | AssertionLabel : Predicate
Predicate
      : Assertion | expression
      | if (expression) Predicate [else Predicate]
expression
      : unary and binary C expressions; constants; and identifiers
identifier:
      system attribute name, user variable, constant,  AssertionLabel
Filter
      Event EventName FilterDescription EventValue
      Status StatusVariable FilterDescription
FilterDescription
      : expression | if(expression) expression [else expression]
EventValue
      : expression | if(expression) expression [else expression]
```

Declarations contain meta-information about the contract as well as type specifications for variables used in the contract. The value of the contract is the logical AND of all assertion values. Assertions may optionally be given a label. If an assertion is labeled, it may be verified (computed) independently of the contract within which it resides. In addition, other assertions may refer to its value using its label. Assertions consist of predicates, which are logical expressions formed using system attributes, customer dependent variables, constants, and arithmetic and logical operators. System attributes and customer dependent variables are set by the measurement plug-ins and the verification interface respectively, and are thus treated as read-only within the language. Filters can be associated with a contract. Filters may be used to compute various kinds of status information (e.g., expected time to repair, trouble ticket information, etc.) about the contract and/or generate events (e.g., notify operators when a contract is not met) when certain conditions are met. Filters and events are defined using a syntax similar to assertions, and can use any of the variables defined as part of the contract.

[1] We use system attributes and system measurements interchangeably. System attributes are abstract or derived measurements computed from element level measurements. Examples are service availability, response time, thruput, utilization, etc.

5 Experience using Conformance

We have tested the Conformance prototype using live measurement data we are collecting from the various XIWT member sites and a large ISP.

Figure 6 shows the experimental setup. The ISP (a large national ISP) monitors its network, the various servers that compose its Email service, and the POP (Point of Presence) sites using active and passive measurements. The measurement data is pushed by the ISP to a measurement station outside the HP firewall. Available measurements include availability and response time measurements from DNS, POP sites, mail servers, and NFS. In addition, measurement stations at HP and several partner sites make periodic network delay and packet-loss measurements by pinging one another. Measurement data is pulled through the HP firewall as shown by measurement-specific plug-ins. In our example scenario, we assumed that the following metrics are specified in the service level agreement of interest:

- **Availability** –Email service is expected to be available 99% of the time as measured over any day. The network is available 99.9% of the time between 8:00 AM and 5:00 PM.
- **Performance** – Email performance is characterized in terms of a) response time < 2.5 seconds when an employee retrieves mail. Network performance is measured by a) round-trip delay < 150 ms and b) packet loss rate < 5% when averaged over daily intervals.
- **Utilization** –The ISP is expected to reserve sufficient capacity at its POP (Point of Presence) so that employees are not denied access to the email service.

The ISP is expected to create daily, weekly, and monthly reports on the overall service compliance and the individual service level metric values.

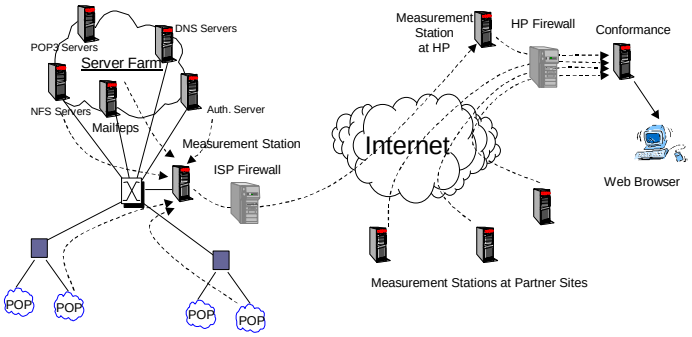

Figure 6. Example Scenario

We now discuss the details of how the email service is modeled and how the contract for email service is monitored. We have constructed similar contracts for the network access services.

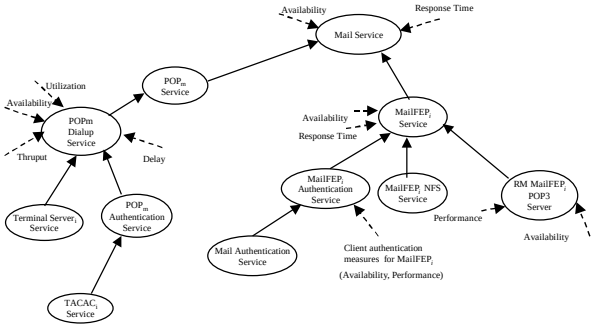

Figure 7 Email Service Model

Figure 7. shows a part of the service model for the email service provided by the ISP. The model is represented as a dependency graph with measurements (indicated by arrows) associated with each node in the graph. The measurements may be made directly (using active tests or passive monitoring) or may be derived from measurements made at other nodes. For example, mail service availability is derived using availability measurements from the POP_m Dialup Service and the $MailFEP_i$ Service. These in turn depend on the availability of the Authentication Service, Terminal $Server_i$ Service, $MailFEP_i$ Authentication Service, and so on[2].

The email service contract is comprised of two service components: the access component (describing the POP and its associated components) and the mail subsystem (describing the mail server and its associated components). A partial description of the contract written in CDL is shown below

```
/* Email Hosting contract template */
Contract Email_System;
Service  Email_Service;
%%
/* POP metrics in SLA- availability, delay, thruput, utilization */
ISP_Access: {
    %popAvailability($popName, …) > $minPopAvailability;
    %popAvgDelay($popName, …) < $maxPopDelay;
    %popThruput($popName, …) > $minPopThruput;
    %popUtilization($dialinServer, …) < $maxPopUtilization;
}
/* Test Mail subsystem for availability and response  times*/
Mail_System: {
    %mailAvailability($mailServer, …) > $minMailAvailability;
    %mailResponseTime($mailServer, …) < $maxMailResponseTime;
}
```

In the contract, *%name* is used as a notation to identify system attributes and *$name* is used to identify customer-dependent parameters to be filled in from the SLA database at the time of the evaluation. The customer parameters define both thresholds (e.g., $minPopThruput) as well as parameters necessary for the measurement system (e.g., $mailServer). Thus, the ISP can check if the email service is meeting SLAs for different customers by filling in the customer specific thresholds (e.g., $minMailAvailability = 99%) and system parameters (e.g., $popName = "Atlanta").

Figure 8 shows a sample report generated by Conformance for the email service contract on a day when problems occurred in the service. This gives the status for the last 24 hours. The aggregate view pie chart shows the percentage of time the contract was compliant vs. non-compliant over the last day. An hourly behavior of the contract is shown in the bar chart, and the compliance percentage for each service component is shown in the ISP Access, and Mail System pie charts. Following other history and detail links gives a historical view of the service, and how the service components are behaving over time.

The report contents and frequencies can be customized depending on what has been agreed upon in the SLA between the provider and the customer. We have generated similar reports for network access using measurements of both inbound and outbound traffic.

In our experience, Conformance scales well. The hierarchical architecture allows domains to be loosely coupled. In addition, because inter-domain interaction takes place using the web, it is easy to replicate Conformance over multiple web servers and share measurement and compliance data using existing servers in the different domains. Finally, because the details of the measurements are hidden from Conformance by the measurement plug-ins, it is easy to add new measurements and couple to different management and measurement systems.

[2] Note that only a small part of the service model is shown in the figure. The leaf nodes on the graph are other services, which have their own service models and associated measurements.

We have found our current implementation of the CDL be sufficient for most SLAs we have encountered. Although the CDL only provides Boolean values for contract compliance, filters defined in contracts can provide multi-valued status information (e.g., warning, critical etc.) prior to the contract being violated. Our current CDL implementation does not support arrays. This causes the system specification in the System Dictionary to become complex for large environments where multiple instances of the same component exist. We are extending our CDL to support arrays to simplify system specification. We believe that both arrays and database support are necessary for a large-scale system.

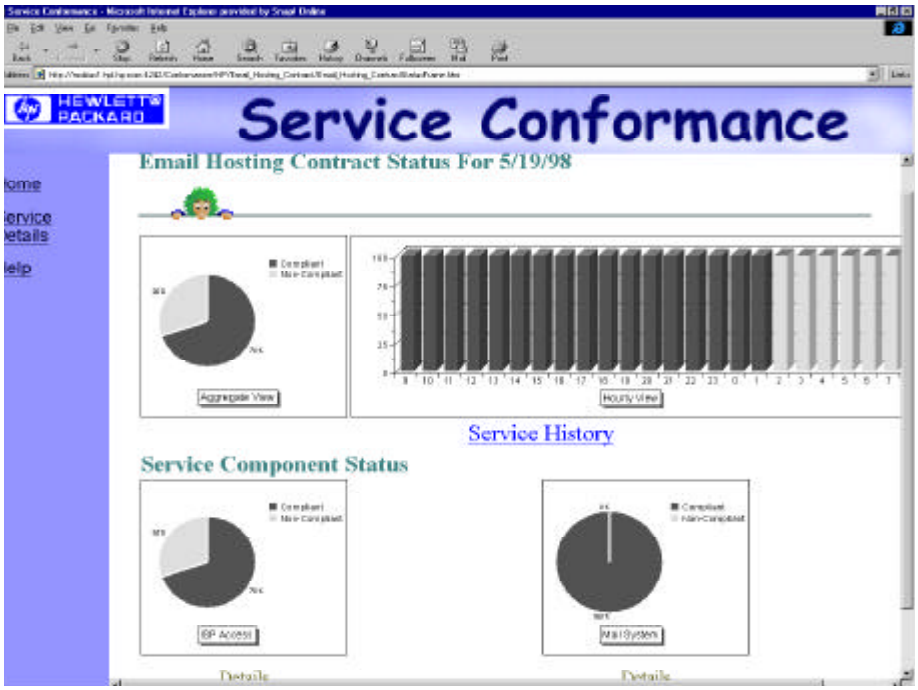

Figure 8: Example of a contract compliance report

6 Summary

Internet services such as e-commerce, web hosting, application hosting, etc., require cooperation and internetworking between multiple organizations, systems and entities while maintaining the confidentiality and privacy of management data considered proprietary by each organization. Currently there are no standard mechanisms to share selective management information between the various service providers. These mechanisms are needed to aid in management and diagnosis of end-to-end services. In particular, service providers are using service level agreements as a means of specifying service level attributes that are offered to their partners and customers. This implies that it is necessary to develop tools and techniques to monitor whether providers are meeting their service level obligations, and to enable providers to manage their infrastructure to those agreements.

In this paper we describe an architecture to share selective management information across multiple business entities. The architecture can be used for automatically measuring, monitoring, verifying, and managing service level agreements for Internet services. The architecture allows specification of attributes that are quantifiable and measurable in a service contract. This allows a service provider to offer verifiable and meaningful service behavior to their customers. Providers can offer customers the capability to automatically verify the current service behavior against the guarantees, by exposing the values of service parameters as agreed upon in the contract.

We also described the design and implementation of *Conformance*, a prototype implementation of this architecture. Conformance is web-based, i.e., uses the standard HTTP protocol, to allow easy inter-domain communication. It isolates the abstractions used in the service level agreements from the details of the service implementation, thus allowing management information to be shared across domain boundaries while hiding the system implementation details.

We have used our implementation to demonstrate how service providers can offer SLA monitoring capabilities to their customers for a number of services including email and network access services.

7 References

1. Bhoj, P., Caswell, D., Chutani, S., Gopal, G., Kosarchyn, M. (1997) Management of new federated services. *Integrated Network Management V.*
2. Thaler, D., Ravishankar, C. An Architecure for Inter-Domain Troubleshooting. *ICCCN '97.*
3. Roscheisen, M., Winograd, Terry. (1997) The FIRM Framework for Interoperable Rights Management. *Forum on Technology-based Intellectual Property Management*.
4. Lewis, D., Bjerring, L., Thorarensen, I. (1996) An Inter-domain Virtual Private Network Management Service. *IEEE/IFIP Network Operations and Management Symposium (NOMS) '96.*
5. Kong, Q., Chen, G., Hussain, R. (1998) A Management Framework for Internet Services. *IEEE.*
6. Irodosek, G., Kaiser, T. (1997) Determining the Availability of Distributed Applications. *Integrated Network Management V.*
7. Infovista&trade Corp., http://www.infovista.com
8. Micromuse, Inc. http://www.micromuse.com
9. VitalSigns Software, Inc. http://www.vitalsigns.com
10. Concord Corp. http://www.concord.com
11. Unified Modeling Language (UML) http://www.rational.com/uml/resources.html
12. Common Information Model (CIM) http://dmtf.org/cim/cimdoc20.PDF
13. Wbem http://wbem.freerange.com