Performance and Service Level Considerations for Distributed Web Applications

Arun Iyengar, Richard King, Heiko Ludwig, and Isabelle Rouvellou
IBM T.J. Watson Research Center
P.O Box 704
Yorktown Heights, NY 10598
{aruni,rpk,hludwig,rouvellou}@us.ibm.com

ABSTRACT

This paper presents an overview of key factors affecting performance of Web sites which may be using Web services protocols. We show how Web sites can be scaled with load balancers sending requests to multiple servers which can be distributed for both improved performance and availability. Caching, load balancing, and replication are critical components for achieving good performance and high availability. We also explain how service level agreements can be used to specify, enforce, and differentiate quality of service within a distributed Web application.

1. INTRODUCTION

Web performance is critically important for electronic commerce and efficient information dissemination. A popular Web site may need to satisfy over a million requests a minute. The infrastructure needed to support highly accessed Web sites is significant. Performance problems may result from a number of factors including insufficient network bandwidth and server CPU capacity. This paper provides an overview of key components which are essential for high performance Web service. These include the use of multiple servers for satisfying different requests concurrently, load balancers, caching, and content distribution networks.

In order to guarantee quality of service, service level agreements (SLA's) may be used which specify the levels of service which are to be maintained between two or more parties. We give examples of using SLA's to specify service levels between Web sites, clients, and content distribution networks.

2. HIGH PERFORMANCE DISTRIBUTED WEB APPLICATIONS

In order to serve Web data efficiently, multiple Web servers are needed. Since requests from different clients are usually independent of each other, several requests can typically be satisfied concurrently, either by a single Web server or by different Web servers. The use of multiple Web servers improves both performance and availability. Since a Web server may fail, having multiple servers allows requests to be distributed to a live server in case a server fails.

Capacity requirements for a Web site may change dramatically over time. Web traffic is typically bursty. Request rates during peak periods may be significantly higher than request rates during quiescent periods. Many Web sites experience peak loads during business hours. Late at night, traffic is considerably lower. Specific events can also cause a Web site to experience high loads. A news Web site might experience high request rates after a major event occurs. A sports Web site might experience high request rates while a major sporting event is going on or right after the event completes. Web sites need to plan for periods of peak capacity, not just average capacity. Therefore, during periods of low request rates, the Web site might have considerable spare capacity which is not used.

Web traffic can also change significantly over time. If a site becomes popular, hits to the site can dramatically increase. The general increase in Web use over time can result in increased demands. As a result, even companies which have not grown substantially have experienced increases in Web traffic which have necessitated substantial increases in hardware.

The overhead for satisfying Web requests varies considerably depending upon the requests. Web sites serve two types of data. *Static data* exists at the time requests are received by servers and are typically stored in files. *Dynamic Data* is created by servers on-the-fly in response to a request. Dynamic data may include information which is rapidly changing. It may also include personalized information which has to be generated uniquely for each client. Dynamic data may be orders of magnitude more expensive to serve than static data. Therefore, even if dynamic data only comprises a fraction of the requests generated at a Web site, the overhead for serving the dynamic data may be the dominant factor in negatively affecting Web performance.

The overhead for serving static data is usually a function of the size of the data. Larger objects consume more network bandwidth to serve than smaller ones. A single HTML page will typically include several image files. In order to

minimize network bandwidth consumption, these image files should be kept small.

Figure 1 shows an example of a distributed Web site. Multiple Web servers are deployed to handle high request rates. The number of Web servers may be scaled appropriately to handle the incoming traffic. The Web site may be geographically dispersed. Having the Web site spread across different geographic regions can improve performance if requests are routed from a client to a Web site which can be accessed with the least latency. Geographical distribution can also result in higher availability. In the event that an entire Web site goes down, requests can be routed to a Web site which is functioning.

Caching is also a critical component for improving Web performance. Caching may take place at several points between the client and the server. A Web browser may cache an HTML page or image file so that subsequent requests for the object can be satisfied without going out to the network.

Caching can also take place within a proxy server. Proxy servers are typically deployed to service clients within administrative and regional domains. Requests from the client first go to the proxy server which sends the request on behalf of the client. The proxy server may be configured with a fire wall. The proxy server masks the IP address of clients it is servicing. Servers only see the IP address of the proxy.

The proxy server can cache frequently requested static data. If several clients are accessing the same cacheable document, then the document only has to be fetched once by the proxy server. Subsequent requests for the document are satisfied from the proxy server's cache.

The cache consistency provided by proxy caches is limited. Objects stored in proxy caches have expiration times associated with them. Once the expiration time has elapsed, the object is considered to be obsolete and should no longer be served. Expiration times only provide a weak level of consistency. In many situations, it is not possible to predict the expiration time of an object in advance. If the expiration time for an object cannot be reliably assigned in advance, the object should not be cached by a proxy. Studies have shown that proxy cache hit rates are generally below 50%. This is not just a result of dynamic, personalized, or other data which are uncacheable.

Requests for cacheable objects not stored in a proxy cache are common. There are a number of reasons why an object may not be in cache. The range of objects that clients behind a proxy server request may be quite large. Some requests are made to objects not previously requested by any client behind the proxy. Other requests are to objects which have expired or been replaced due to cache overflow.

If a cache fails or is shut down, the contents of its volatile memory will be lost. Hit rates may be low during the period when the cache is brought to a warm state. In order to minimize start-up costs after a failure or shutdown, frequently requested objects should be backed up in persistent storage.

Caching can also take place within a content distribution network (CDN). CDN's cache content at edges of the network to bring content closer to clients. By bringing content closer to clients, latency for accessing objects is reduced. Server load is also reduced by off loading some of the request content. Since CDN's generally don't store dynamic content, however, server load will probably not be reduced when the bottleneck is the CPU overhead for creating dynamic content.

Dynamic and personalized Web pages introduce complications for caching. One technique for caching at least parts of dynamic and personalized pages is to define them as being composed of multiple fragments [2]. In this approach, a Web page is defined as being composed of simpler fragments. Some fragments may be cacheable, while others are not. Highly dynamic and personalized parts of Web pages would typically not be cacheable, while more static parts of the Web page would be cacheable. Using this approach, a remote cache stores some fragments, and fetches others from the server. The Web page is assembled by the cache.

Web applications are evolving from "traditional" business-to-client applications where end users predominantly read and browse mostly static data. Dynamic content is becoming increasingly important, and many Web applications are more accurately described as "service providers" (beyond data providers). Moreover, these services can be combined to form complex processes crossing enterprise boundaries. Service-oriented computing is emerging as a new model, which combines object-oriented and messaging principles, as well as Web technologies. Web services, for example, are applications that are described, published, and accessed over the Web using open XML standards. They promote a service-oriented computing model where an application exposes both its functionality and its mappings to available communication protocols using the Web Services Description Language (WSDL, [3]). This description can be discovered by client applications using service registries in order for the client to then use the service by means of XML messaging. As Web services reach their full potential, new services will emerge that are based on compositions of other Web services and specified using inter-operable business process standards. Traditional techniques to improve Web performance such as static data caching or load balancing need to be augmented with new ones targeted toward this more dynamic, service oriented environment. Below, we discuss a number of techniques that could be used to distribute and replicate business logic across multiple servers.

A number of people have proposed moving processing functions typically done by servers into the network to execute within the network, such as on proxy caches [1]. It is feasible to perform functions such as logging user accesses and access permission checking within the network. In the context of active network research, techniques such as code capsules have been explored to allow application-specific data processing in network nodes as well as rapid deployment of new network protocol implementations [4]. The code being distributed in this work is typically concerned with relatively low level networking functions such as congestion control.

By taking advantage of commonalities in operations involved in a particular application, we can distribute logic as well

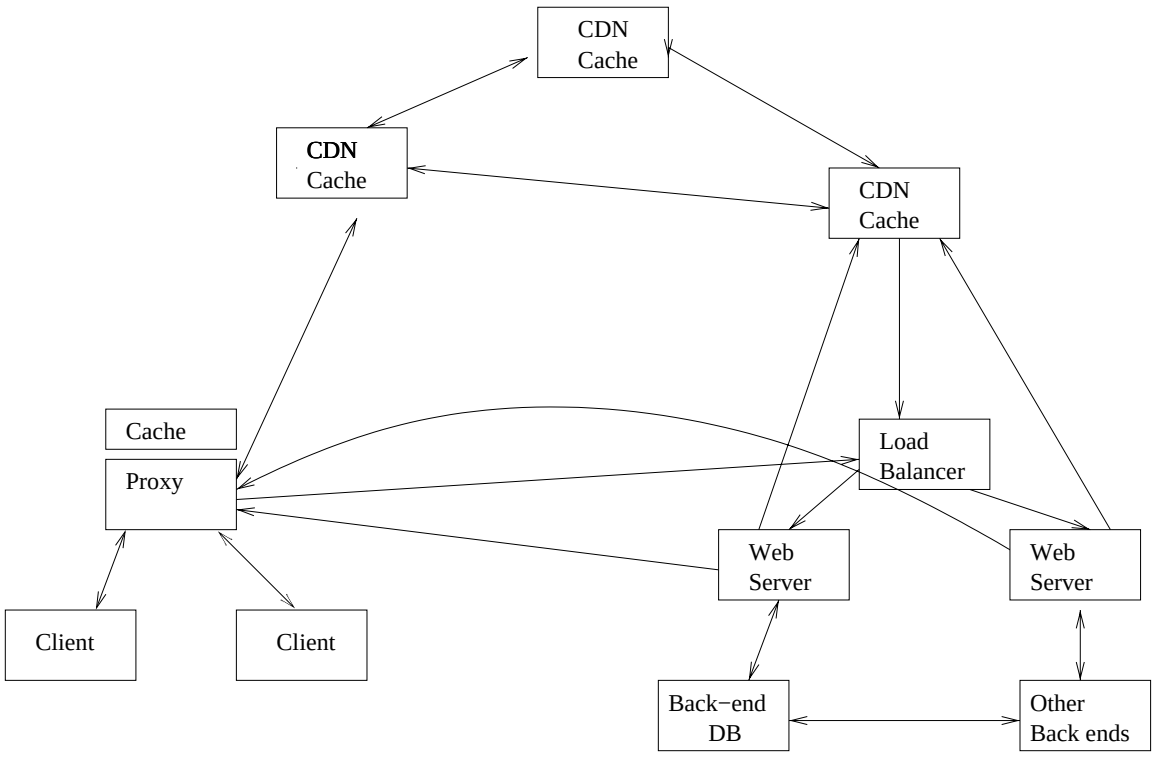

Figure 1: A distributed architecture for serving Web content.

as data. For example, in [5], both data and logic are distributed across a network to improve both availability and performance of an on-line book store based on the TPC-W benchmark. Distributed objects are created based on application-specific semantics to manage shared information using simple but effective consistency models. In [9], a similar problem is addressed by the Globe system, a framework for distributed computing in wide area networks based on distributed shared objects.

An area of future research which we are interested in is to to use aspect-oriented software development techniques [8] to systematically build smart distributed objects from actual objects to allow a Web application to be distributed across a network. This entails the identification of relevant semantic patterns (e.g., one-to-many scatter semantics) which could then be "aspect-ized" using aspect-oriented approaches. The aspects would then be available as composable reusable units for use in the building of smart distributed objects.

3. SERVICE LEVEL MANAGEMENT

An SLA defines the relationship between a service provider and its customer with respect to quality and performance properties of a service. This typically includes the definition of the metrics that the SLA is based on, e.g., availability or response time, the definition of the service level objectives related to the metrics, and the specification of the effects of meeting or violating service level objectives, e.g., notifications or penalties.

SLAs are the basis of *service level management* of both service providers and customers. Service providers typically have service level agreements with multiple customers that often rely on the same resources, e.g., hosts running Web servers and back-end databases. When there are high loads on those resources, service providers have to decide how to allocate resource capacity to SLAs, e.g., by maintaining service levels that just avoid penalties or by prioritizing requests according to SLAs that foresee a high penalty in case of violation. If SLA compliance problems increase across the SLA portfolio of a service provider, it may be cost effective to add new resources.

Service customers monitor SLA compliance of their service providers. In the case of repeated non-compliance, they may want to change a service provider. They may also discover that they are using a service beyond what has been agreed on in the SLA and hence are receiving poor quality service. This would be the case, for example, if the SLA guarantees response times of less than 3 seconds for up to 500 requests per minute but actually 1000 requests are issued. In this case, the customer may want to upgrade the SLA to cover the volume that is used.

The basis for service level management is an SLA compliance monitoring infrastructure. The compliance monitoring infrastructure collects *resource metrics* by accessing instrumentation of devices and applications, or by probing. For Web servers, probing is a widely used method to measure availability and response time, and is also offered as a service by independent organizations. The collected resource metrics usually will be aggregated into high-level metrics that are relevant to the service level objectives. Finally, these

service level objectives evaluated and alerts will be raised in the case of non-compliance or other defined conditions, e.g., a trend in recent measurements suggesting future non-compliance.

The Web Service Level Agreement (WSLA) framework [6] is a compliance monitoring framework that facilitates distributed collection and aggregation of metrics, distributed service level objective evaluation, as well as the use of external parties to support the compliance monitoring process. The architecture of the framework is illustrated in figure 2.

The framework provides two types of components: measurement services and condition evaluators. A measurement service gathers resource metrics and computes high-level metrics. A condition evaluator receives values of high-level metrics and determines if the service level objectives are being met. This division of labor allows an SLA to be broken into parts, with different parties performing only a portion of the overall task of determining SLA compliance, based on that portion of the SLA deployed to them.

For example, a client may wish to hire a third party to gather information on the quality of service being provided. Although the client is willing to let this third party (3rd Party MS) know what measurements are wanted, it is not interested in divulging what constitutes a good measurement rather than a bad one. This is accommodated, as we see in the figure, by having a third party run only a measurement service, leaving the work of condition evaluation in the hands of the customer or, from the service provider's point of view, commissioning another 3rd party (3rd Party CE) for this task.

In order for the WSLA framework to be usable by different parties, the SLA must be expressed in a common language. It must be machine readable, it must describe the parties involved in this agreement, and it has to indicate what work each of them will be doing. There also need to be sublanguages for measurement services and for condition evaluators so that each party can be informed of only that part of the SLA relevant to that one party. Most important is recognition that the SLA is shared between different organizations, requiring a standard language whose meaning is understood by all of the parties to the agreement. The WSLA language [7] is a common language for SLAs that fits the model of compliance monitoring outlined and can be split into parts for individual 3rd parties.

Now, the gathering of resource metrics is inherently system specific. In the figure we see the Provider's measurement service drawing information directly from its own servers, while the 3rd Party's measurement service relies on information derived from dummy, probe requests. Although the mechanisms for getting the information is completely different, the result is the values of resource metrics mentioned in the SLA. A general-purpose WSLA framework must therefore provide means for the attachment of customized mechanisms for gathering such data.

The management systems used by the Customer and the Provider are different, depending on the particular Web server and back-end systems being used by the provider and the particular system used to manage contractual relationships by the customer. Hence, notification of SLA violations requires being customized for those systems having completely different logging and notification protocols and requirements. Rather than encumber the WSLA framework with such details, it is better to provide a fixed notification method to which customized notifiers can be attached. These notifiers are then free to take the information provided by the WSLA framework and transform it into the log requests or management service invocations appropriate to that particular system.

The response time and throughput measures already mentioned are typical of the kind of high-level metrics that would be important in an SLA. Availability is another and is, in fact, the only high-level metric mentioned in many current SLA's.

As an example of how resource metrics are turned into high-level metrics, let's concentrate on throughput, as similar arguments can be made about the others. Systems don't actually record throughput; they usually just count the incoming (or outgoing) transactions and allow that count to be observed. To compute throughput, this transaction counter can be observed periodically, say once per minute, the difference in successive readings taken, and that result can be used as the throughput. Now, this may not be the right time scale, so adjustment of the observation schedule may be needed. Or smoothing over several observations of the low-level counter may be desirable, in which case buffering over a window of past readings is needed. It is these kinds of variations in the precise definition of a high-level metric that indicate the need for an SLA language and a general-purpose framework for compliance monitoring.

4. SERVICE LEVEL MANAGEMENT IN A DISTRIBUTED WEB ENVIRONMENT

To see how service-level management would apply to a distributed Web environment, let's redraw our original picture. In figure 3 we can see the organizational boundaries, the agreements between the organizations, the measurement services and condition evaluators that monitor compliance with the agreements, and the flow of information between those entities.

The primary SLA is between the Customer and the Provider. Although, for example, the response times experienced by the Customer will be determined largely by the behavior of the CDN, that is not mentioned in this SLA as the CDN is a service subcontracted by the Provider. That relationship is therefore the subject of a separate SLA between the Provider and the CDN. The Customer-Provider SLA can then concentrate on the quality of service as perceived by the Customer, and on how that will be measured, rather than on the internal details of how the Provider's systems are arranged.

For the Customer-Provider SLA, measurements will be taken by the Third Party, which will issue fake requests to determine the responsiveness of the Provider's system. This is a usual provision in many of today's service level agreements, and companies such as Keynote provide such third party services. These resource metrics (like individual response

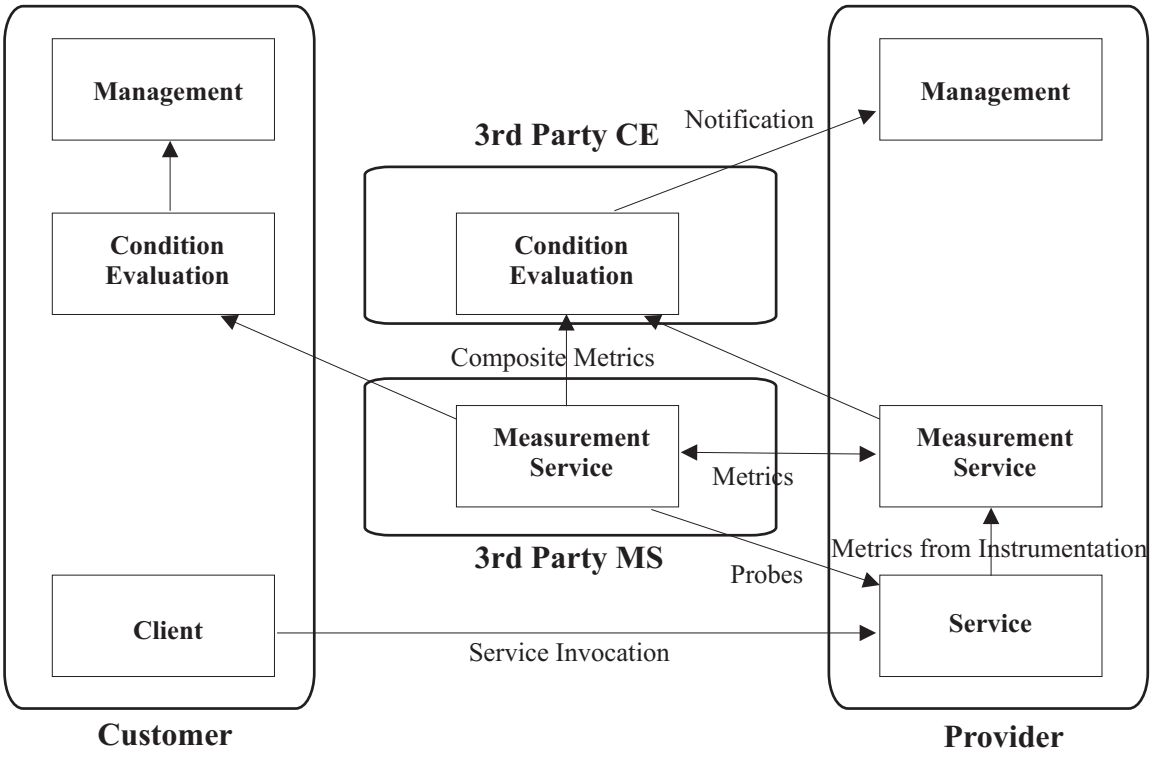

Figure 2: WSLA compliance monitoring architecture.

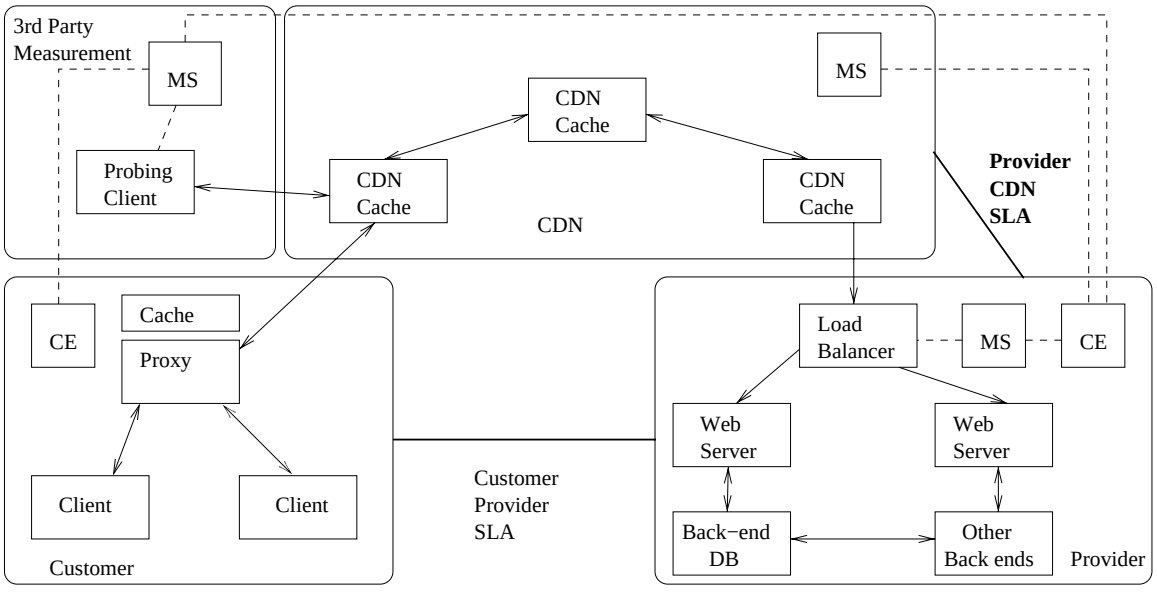

Figure 3: WSLA compliance monitoring scenario.

times) will be gathered in the Third Party's measurement service, transformed into high-level metrics (like average response time), and passed to the Customer's condition evaluator, which notifies the Customer when its SLA with the Provider is not being met.

At the same time, these same measurements are of interest to the Provider, as indications of how well the CDN is fulfilling its SLA with the Provider. Therefore, the Third Party is also included in the Provider-CDN SLA so that the Provider can benefit from that same information. Of course, the Provider and the CDN both have detailed knowledge of their own systems, and have their own measurement services for gathering resource metrics from internal interfaces available only to themselves. The high-level metrics derived from all of those sources go to the Provider's condition evaluator, which warns the Provider of problems within its own servers and with its relationship with the CDN.

Based on information about SLA compliance, a service provider can analyse the root cause of the problem and then take corrective measures such as adding servers to its own back end or upgrade its SLA with the CDN to increase the cache size or the geographies being supported.

5. CONCLUSION

This paper has presented an overview of factors affecting the performance of distributed Web sites. Performance and availability are enhanced by using multiple Web servers which can be geographically distributed. Requests are routed to different servers using load balancers.

Caches are critically important for improving performance. Proxy caches and content distribution networks are commonly used to bring content closer to clients. While caching is usually restricted to static content, fragment-based caching allows static parts of dynamic and personalized Web pages to be cached.

We also discussed service level agreements and the Web Service Level Agreement (WSLA) compliance monitoring framework which can monitor SLA compliance in a distributed way using external, third party participants. This monitoring framework allows service providers to tailor their resource commitments to the demands of their customers; SLAs may be associated with content distribution networks. This approach promises to cut costs for excess capacity within service providers. Furthermore, customers can have their SLAs monitored independently by third parties without relying on service provider metrics. Our framework allows service providers to pursue a similar approach not only for content distribution but also for other functions such as load balancing.

6. REFERENCES

[1] P. Cao, J. Zhang, and K. Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proceedings of Middleware 1998*, 1998.

[2] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. A Publishing System for Efficiently Creating Dynamic Web Content. In *Proceedings of IEEE INFOCOM 2000*, March 2000.

[3] E. Christensen et al. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl.

[4] D. Decasper and B. Plattner. DAN: Distributed Code Caching for Active Networks. In *Proceedings of IEEE INFOCOM'98*, 1998.

[5] L. Gao, M. Dahlin, A. Nayate, J. Zheng, and A. Iyengar. Application Specific Data Replication for Edge Services. In *Proceedings of the 2003 International World Wide Web Conference (WWW2003)*, May 2003.

[6] A. Keller and H. Ludwig. The WSLA Framework - Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and System Management - Special Issue on E-Business Management*, 11(1):57–81, 2003.

[7] H. Ludwig, A. Keller, A. Dan, R. King, and Richard Franck. A Service Level Agreement Language for Dynamic Electronic Services. *Journal of Electronic Commerce Research*, 3(1):43–59, 2003.

[8] P. Tarr et al. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering (ICSE '99)*, May 1999.

[9] M. van Steen, P. Homburg, and A. Tanenbaum. Globe: A Wide-Area Distributed System. Technical report, Vrije Universiteit, March 1999.