Modeling of Service-Level Agreements for Composed Services*

David Daly[1], Gautam Kar[2], and William H. Sanders[1]

[1] Center for Reliable and High-Performance Computing,
Coordinated Science Laboratory and
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign,
Urbana, Illinois 61801, USA
{ddaly, whs}@crhc.uiuc.edu,
http://www.crhc.uiuc.edu/PERFORM
[2] IBM T.J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598
gkar@us.ibm.com

Abstract. As Web services are increasingly accepted and used, the next step for them is the development of hierarchical and distributed services that can perform more complex tasks. In this paper, we focus on how to develop guarantees for the performance of an aggregate service based on the guarantees provided by the lower-level services. In particular, we demonstrate the problem with an example of an e-commerce Web site implemented using Web services. The example is based on the Transaction Processing Performance Council (TPC) TPC-W Benchmark [8], which specifies an online store complete with a description of all the functionality of the site as well as a description of how customers use the site. We develop models of the site's performance based on the performance of two sub-services. The model's results are compared to experimental data and are used to predict the performance of the system under varying conditions.

1 Introduction

Web services are increasingly being referred to as the future of outsourcing on the Internet, because they allow remote services to be discovered and accessed in a uniform manner to execute some functionality. The infrastructure for Web services is already being developed in the form of open standards such as SOAP,

* This material is based upon work supported in part by the National Science Foundation under Grant No. 9975019 and IBM. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or IBM. This work was done in part while Mr. Daly was an intern at IBM T.J. Watson Research Center.

M. Feridun et al. (Eds.): DSOM 2002, LNCS 2506, pp. 4–15, 2002.

UDDI, and WSDL, as well as other closed standards. A Web-service customer can query a UDDI [10] (Universal Description, Discovery, Integration) server to find needed services, and access WSDL [2] (Web Service Description Language) descriptions of the services using the SOAP [1] (Simple Object Access Protocol) protocol. The customer and service provider negotiate a contract, including service-level agreements (SLAs), once the customer finds the appropriate service.

The development of outsourced Web services allows service providers to be more specialized and efficient, and to provide improved, more flexible services. A logical result of this outsourcing is the development of hierarchical services (which are themselves made of outsourced services) as well as service aggregators that can develop a complete Web service for a customer using outsourced services. With little overhead, a service aggregator would be able to quickly develop and deliver a service for a customer; It would contract only for the needed levels of service, and would increase those levels as needed. We focus on the modeling of such services to determine the service levels that can be guaranteed.

2 Problem Overview

Hierarchical and aggregated services are also Web services, and require that contracts, including SLAs, be negotiated. SLAs stipulate minimum standards to be provided by the service and usage constraints for the customer of the service. The agreements also generally include penalties if the service levels do not meet the guarantees (that is, if there is an SLA violation). It is a relatively straightforward (although not necessarily easy) problem to determine what levels of service can be guaranteed when the service is provided entirely by one entity (no outsourcing of subcomponents); the prediction of performance of individual services has been examined in many areas. However, it is more difficult to determine the level of service that can be offered for a service composed of multiple services. The service provider may know only the guarantees offered in the SLAs of the component services, and therefore will need a method to compute the overall SLA of the composite service.

Thus, the problem is to develop contracts between the customer, the service integrator, and all of the service providers. The service integrator guarantees overall performance to the customer, while the service providers guarantee the performance of their services to the service integrator. The problem is complicated by the fact that the service level of the composite service may not be a simple combination of the service levels of the sub-services. This situation requires a simultaneous analysis of all the relevant outsourced services, which is the focus of this paper.

Before we can address the problem of relating SLA terms, we first must understand the type of guarantees offered by an SLA. Providers commonly guarantee that service will be completed within a certain time, a certain percentage of the time, when the load is below a certain value. This will normally be worded as "X% of requests respond in under Y seconds, when the load is less than Z requests per second." We used SLAs of that form for the component services.

2.1 Focus Area: E-commerce and TPC-W

For this paper, we focused on a situation in which a service integrator implements an e-commerce Web site for a client. The integrator may use a combination of traditional service providers and outsourced Web services to implement the e-commerce site, providing the integrator with the flexibility to scale the services as needed with low overhead. Whether the service integrator is an external vendor or an internal organization is immaterial. It will still need to provide performance guarantees to the client.

The e-commerce example we specifically examine is the TPC-W benchmark [8], which was developed by the Transaction Processing Performance Council (TPC) [9]. It is a benchmark based on an online bookstore. Users may search for books, view bestsellers, track orders, and perform other functions. All of the pages are dynamically generated using an application server, and all product and customer data are stored in a back-end database. The TPC-W benchmark, therefore, requires two main services: the application server and the back-end database. While the benchmark is intended to test unified system offerings from a vendor, there is no reason why both services could not be outsourced.

The standard SLA guarantee based on the percent of satisfied requests, while satisfactory for simple services, is not sufficient for an e-commerce site, such as the example one. A shift towards business metrics is necessary to properly meet the requirements of the client. The client does not inherently care about response time, but about the satisfaction of the customers using the site. Ultimately, the client is concerned with revenue and profit, but the service provider cannot make guarantees on revenue and profit, as there are many factors other than service level that influence those metrics. However, if a customer becomes dissatisfied with the site and leaves, the client loses potential revenue. Therefore, we propose to relate the business metrics that a client is interested in to the service-related metrics that the provider is able to measure and report: the number of customers who leave a transaction prematurely.

Therefore, we suggest SLA terms for an e-commerce site of the form "less than X% of the customers leave the site prematurely because of the service level." The fraction of customers who leave the site prematurely will be dependent on the response time and the service levels of the component services, but cannot be represented by a single response time guarantee. We do use the original SLA form presented earlier for the component services, since they are simple services, but the more sophisticated SLA is required for the complete e-commerce site.

2.2 Related Work

Menascé et al. [6] have performed some relevant related work on modeling e-commerce systems. They use revenue as the key metric, and determine what effect several options have on revenue. In [5] Menascé and Almeida develop several queuing models of e-commerce systems to determine the resources needed to meet the demands on the system. Adjustments are made for variability in workload, and for multiple classes of requests. The demand for the system is

generated using a Customer Behavior Model Graph (CBMG) that is solved to determine arrival rates in the queuing systems. The authors have extended the work to make it business-focused by concentrating on revenue, with metrics such as revenue throughput and potential lost revenue [6]. The work assumes that all resources (application server, database, and so forth) are controlled by the company hosting the e-commerce site. Based on that assumption, it is valid to focus directly on the revenue and profits of the site.

Our work deals with similar systems, but in the context of outsourced Web services. The use of outsourced Web services invalidates the assumption that all resources are under the direct control of any one service provider. In addition, since a Web service aggregator is merely developing and implementing the e-commerce site for the company, and does not control the products offered by the site, it is not reasonable to expect revenue guarantees from the aggregator. Therefore, instead of focusing on revenue, we measure other factors that impact revenue, and can be controlled by the aggregator.

3 E-commerce SLA Models

To determine the SLA guarantees that a service integrator can offer to a customer based on the SLA guarantees of outsourced services, we develop a model of the service using many submodels. The model has two major components that it uses to determine SLA guarantees. The first is the workload model, which models the load applied to the system. It incorporates the behavior of the users to determine how many requests are made. In addition, since our SLA for an e-commerce site is based on user behavior (how many users leave the site prematurely because of poor service), the workload model must explicitly model users leaving the site both prematurely and after completion of normal activity. The second component is the system model. The system model captures the performance of the services as they process the user requests.

The workload model needs to determine the load offered to the services, and must also predict how many customers will leave the site prematurely. We must therefor understand what makes a user leave a Web site. For that reason, the workload model includes models of several users accessing the services. We have used a simple model of user behavior in the workload model. The users wait a certain amount of time for each page. If the page takes longer than the allowed time to load, the user attempts to reload the page. After he/she attempts to reload a page a certain number of times, he/she becomes frustrated and leaves the site. We model this by tracking how many times a user reloads a page, and when the number of reloads gets above a preset threshold, the modeled user leaves the site. We could construct a slightly more complicated model in which the user leaves the site only if he/she must retry too many pages in a certain span of time. That scenario is more complex and will be dealt with in future research.

We need a model of the system to use with the workload model. The system is made up of the individual services. The model of the services has a latency

and a throughput component to determine the total delay experienced by a user request. The latency component in the model represents the network latency in sending the request to the server and getting the response from the server. All the network latencies are combined into an aggregate latency for each service, which is not affected by the load on the server. If the local network itself is expected to be a bottleneck, it should also be modeled as a service.

We represent the latency as a constant time delay. This corresponds well to a low network load condition, in which all requests of the same size take the same amount of time to traverse the network. However, if the network is the Internet, it may experience local bottlenecks and varying delays. We do not attempt to account for that factor at this time.

The second factor in the total delay a request experiences in accessing a service (over and above the latency) is the service time. The service time is the total time to process a request, once that request has arrived at the service. For example, a number of factors affect the service time of a request, including the size of the request, the speed of the service, and other requests at the service. If the service exhibits parallelism, it can process multiple requests at the same time with no degradation of service to the requests. The number of requests that can be processed at the same time with no degradation of service to each request is the degree of parallelism of the service. If the number of requests is greater than the degree of parallelism, then all requests are processed at the same time, and they all experience a slowdown in processing.

The service time, combined with the level of parallelism, determines the throughput of the service. The throughput is important to the system operator, as the operators will want to process as many requests as possible on the given hardware. However, the user ultimately does not care about the throughput of the service, only the delay experienced in accessing it. The parallelism and service time allow us to determine the delay, which we would be unable to compute solely from the throughput.

3.1 Parameters Based on SLA Values

The server processing delay is selected to match the SLA guarantee on loss for that service if no more detailed information is available. Recall that the service SLA was defined in Section 2 to have the form "no more than X% of requests take longer than Y seconds to complete when the load is less than Z requests per second." Values for the service delay and for the parallelism of the service need to be determined from those SLA values.

We select the average service delay to meet the SLA guarantee on an unloaded machine. This delay is specified by the delay distribution of the service; the specification includes both the type of the delay distribution (e.g., normal or negative-exponential) and any parameters used to describe the particular distribution type. We use a negative-exponential delay distribution if no other information is available. The negative-exponential distribution requires only the average service time as a parameter. It is a simple matter to determine the parameter for a negative exponential distribution such that X% of the time a

sample value will be greater than Y. The service time of a request on a loaded server is more complicated, since the request may need to compete for resources, leading to longer service delays. The effect that load has on the request is reflected in the degree of parallelism of the service; by setting the parallelism to Z times the service delay we can select the parallelism such that the server can process more than Z requests per second without experiencing any slowdown.

4 Experimental Validation

We demonstrate the models and ideas developed in the previous section by simulating experiments in the Möbius tool [3,4] and running experiments on an experimental TPC-W configuration to determine the lost user rate for the e-commerce system represented in TPC-W. We do not explicitly develop SLAs for the two services, but instead use the models developed for Web services and perform measurement on experiments to determine the model parameters (as suggested in the previous section). Therefore, our experiment is unusually detailed, and should demonstrate the accuracy of the models. In addition, the measurements show how the SLA guarantees for the component services could be determined.

4.1 Simulation Environment: Möbius

The models were simulated using the Möbius tool. Möbius is a multi-formalism, multi-solution extensible modeling tool for discrete event stochastic models [3, 4]. Using Möbius, a user can develop models of parts of a system using different formalisms (or ways of describing a model) and combine those models to form a complete model. For this study we used the Stochastic Activity Network (SAN) [7] formalism, because of its generality. SANs consist of places (represented by circles), which contain tokens; activities (represented by bars), which remove and place tokens in places; and gates (represented by triangles), which control the behavior of activities, allowing for more complex behavior.

Figure 1 shows the SAN model of a user accessing the home page of the site. The process starts when a token is placed in Home by another submodel. (That submodel is composed with other submodels to form the complete model. Composition is explained below.) When the user generates requests, the `Request` activity fires, removing the token from `Home` and placing one token apiece in `Home_in` and `Req_in_Prog`. The user waits for a token to be placed in place `Home_out`, which represents a response. When the token is placed in `Home_out`, that token and the token in `Req_in_Prog` are removed, and a token is placed in `viewing`. After that occurs, some time will pass (representing the time the user spends reading the page) before the `Done_Viewing` activity fires, removing the token from `viewing` and putting it in a place that determines which page to visit next. Alternatively, if the response to the request takes too long, the activity `Timeout` will fire, and the user will retry the request. The `Drain_Lost` activity ensures that the lost request will eventually be removed from the system.

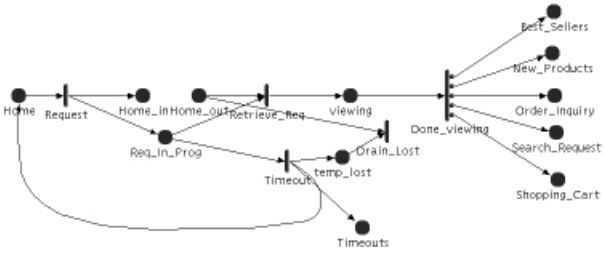

Fig. 1. SAN Model of a User Accessing the Home Page

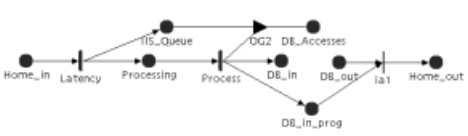

Fig. 2. SAN Model of Application Server

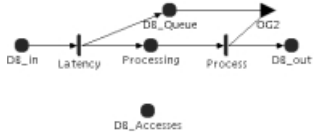

Fig. 3. SAN Model of DB Access for Home Page

The model does not explain how a token goes from `Home_in` to `Home_out`. The home_service model, shown in Fig. 2, controls that. The `Home_in` and `Home_out` places will be shared between the two models, so both models will always have the same number of tokens in each place. The `Latency` activity will remove a token from the `Home_in` place and place one token a piece in `Processing` and `IIS_Queue`. The delay of the `Latency` activity represents the network latency experienced by the request. The `IIS_Queue` is shared by all the services that use the application server. The processing time of the home page is scaled by the `IIS_Queue` value combined with the degree of parallelism in the server. The home page also requires a DB access, and the time for this access is modeled after the application server processing time. A token is placed in `DB_in`, and when the DB request is done, a token is placed in`DB_out`.

Figure 3 shows the model of the DB access. It is similar to the home page access. When the token is placed in `DB_in`, there is a delay for the firing of `Latency`, and then some processing time, which is scaled by the number of other requests currently in the DB. Similarly, there are models that represent the user access to each of the pages in the site and the processing of those requests. In our earlier description of the user behavior model, we stated that the user leaves if too many requests are retried. That is modeled in a separate model that we do not describe here because of space consideration. When a user retries too many pages, he/she ends the session prematurely. A user session can also end normally. According to the TPC-W definition, a session ends on the home page request after a random timer expires. This behavior is included in our model.

Möbius allows multiple models to be composed together, with certain places held in common or shared through the Rep/Join formalism. A join node combines multiple different models, while a rep node creates multiple replications of one model. In both cases, some places are held in common between the submodels.

Our models are joined together, sharing all identically named places, to create a model of one user accessing the site. The model is then replicated to represent a population of users accessing the site. The replicas in the final composed model share only specific places, such as IIS_queue and DB_queue.

The measure of interest for this model is the percentage of user sessions that end prematurely. Therefore, measures are defined to determine the number of sessions that end prematurely, and the total number of sessions. The ratio of those two numbers is the user loss rate.

4.2 Experimental Environment: TPC-W Running in Lab

In our lab we set up an application server and a database to run the TPC-W benchmark on a local network. The application server ran using IIS and Jakarta on an IBM RS/6000 workstation, while the back-end database resided on an AIX box, using DB2. The two servers and the client were connected using 100 Mbit Ethernet. The client ran under Linux on a Pentium III workstation.

In TPC-W, users wait for each request to complete and end sessions normally. We adjusted our TPC-W client emulators to retry requests if too much time elapses, and to leave the site if the client has to retry too many requests. With that adjustment, the model should match the behavior of the experimental setup after one additional step: the model needs to be parameterized to match the experimental system. Since we did not have actual SLAs from which we could determine the parameters, we instead attempted to experimentally determine the response times and parallelism of the services. We calibrated these values by performing experiments without user timeouts; in other words, the users would never retry requests. Single-user experiments were run to determine the response time of the servers, while multiple-user experiments were run to determine the effect of load on the services, and therefore the parallelism. A service provider would perform similar experiments to determine the SLA terms that could be offered for a service. We analyzed each page on the TPC-W, since each one makes different demands on the outsourced services.

5 Results

We start with the calibration results. Calibration was needed to determine the service delay and parallelism of the two services (application server and database server), as well as the size of the requests made by each page. The step can be thought of as determining what terms could be offered for an SLA on those services, and translating them into the needed parameters. Indeed, this calibration was equivalent to having SLA terms that were very accurate plus the distribution information.

We had to calibrate the pages individually. Each page had multiple database accesses and varying usage of the application server. Figure 4 shows the inverse cumulative distribution function for the delay for accessing the buy confirm page and the home page. The simulated results closely reflect the experimental results.

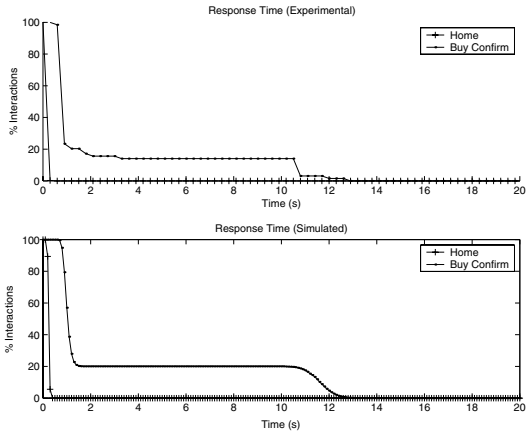

Fig. 4. Inverse Cumulative Delay Distributions for Page Accesses

The services had a low variance on the completely unloaded systems; we accommodated that by using an Erlang distribution instead of a negative exponential for those services. The Buy Confirm page had a large outlying probability density; it was around 10 seconds for our experiments. To accommodate that we adjusted the DB access to occasionally require larger amounts of resources. The other pages performed similarly well, with the Buy Confirm page being one of the slowest pages, and the Home page being one of the quickest pages.

A service integrator would determine how many requests each page made to the outsourced services and the size of the requests, and would then determine the model parameters by combining that information with the performance guarantees offered by the service providers for the outsourced services. We determined relative request sizes and performance for the service based on measurements of the delays of the pages and an understanding of the requests generated for each page.

5.1 Validation of Model Predictions

With the models calibrated, we ran complete experiments to compare the loss rate of the modeled and experimental systems. We found that the two systems gave similar overall loss rates and component loss rates, as shown in Table 1. The table shows user retries and the number of lost users when there are 30, 45, or 60 users with users willing to wait 10, 12, or 15 seconds. All experiments were for 15 minutes after a 90-second warmup period. The simulations solved the model several times to generate 95% confidence intervals compared to the experiments that were solved once. We note two things about the results: 1) at lower loss rates, our model reports more losses and timeouts, and 2) at higher loss rates, there appears to be a higher probability that a retried request will take too long, leading to loss that was not captured by our models. Therefore we expect our models to be conservative (to overestimate loss) at low loss rates. We might be

able to account for the higher loss rate when there are more retried requests by making retried requests have a higher demand on the services, especially since requests that need to be retried are likely larger than average to begin with.

Table 1. Simulation and Experimental Results

Number of Users	Timeout Length	10		12		15	
		Exper.	Simulation	Exper.	Simulation	Exper.	Simulation
30	Timeouts	12	9.65 ± 0.55	0	2.30 ± 0.27	0	0.46 ± 0.10
	Loss	7	1.76 ± 0.18	0	0.20 ± 0.07	0	0.02 ± 0.02
45	Timeouts	13	15.19 ± 0.81	4	4.35 ± 0.40	0	1.47 ± 0.20
	Loss	7	3.09 ± 0.28	0	0.52 ± 0.12	0	0.12 ± 0.05
60	Timeouts	26	20.92 ± 1.21	7	7.67 ± 0.71	1	3.49 ± 0.40
	Loss	13	4.68 ± 0.47	1	2.30 ± 0.27	0	0.42 ± 0.10

From the results, we could determine the SLA that could be offered by the service integrator, based on the amount of time that a user will wait for a page. For instance, if a user could be expected to wait 12 seconds, the SI could guarantee that there would be less than 1% loss when the load is less than 60 users.

One problem we discovered was that the system could become unstable at high loss rates. The TPC-W benchmark starts a new user session immediately when another one ends, in order to maintain a constant number of users. That can lead to an increased load when loss is considered. Normally, a user would leave, causing the load and loss rate to drop; but in our experiment, the new user will also keep retrying requests, increasing the load instead of decreasing it.

5.2 Analyzing Results from Varying Parameters in the Simulation Models

Once we verified that the model performed well for the base case, we conducted some studies using the model to better understand the dynamics of the system and their ramifications for the performance that could be guaranteed. Some of the studies could not have been done with the experimental setup, while others would have been time-consuming. The studies focused on ways to lower the overall loss rate, in order to improve the service guarantees that could be offered.

The first two studies focused on the user model and on determining the effect of timeout value and loss threshold on overall loss rate. We varied the two parameters separately in two studies. Figure 5 shows that increasing the timeout value did decrease the loss rate, and increasing the timeout value to 14 seconds would ensure a loss of no more than 1%. Similarly, Fig. 6 shows that if the users are willing to retry three requests before leaving, instead of one, the loss rate also drops below 1%.

However, since the timeout length and the loss threshold are given values, we cannot change them. Instead, we have to focus on ways to speed up the site to

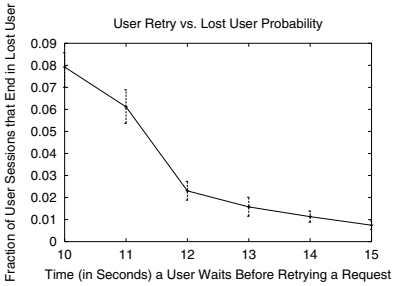

Fig. 5. Loss Rate as a Function of How Long a User Will Wait for a Request

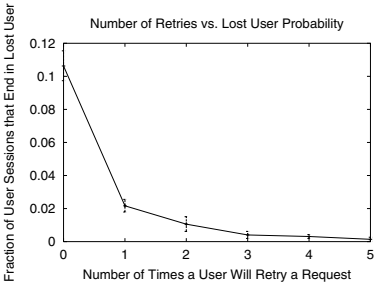

Fig. 6. Loss Rate as a Function of How Many Times a User is Willing to Retry More Requests

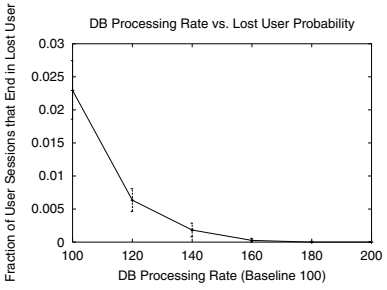

Fig. 7. Loss Rate as a Function of DB Performance

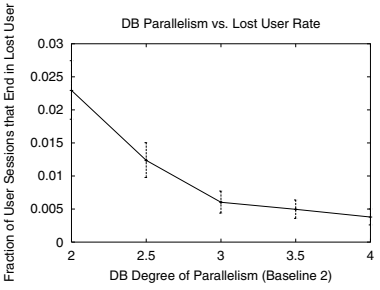

Fig. 8. Loss Rate as a Function of DB Parallelism

lower the loss rate. Since the two services we were evaluating were the application server and the back-end database, we also varied the performance of those two services. Minimal speedup was observed from increasing the application server performance, but performance degradation was observed if the application server is slowed down. However, increasing the performance of the database has a dramatic effect on the overall loss rate, as shown in Fig. 7. Therefore, the database is the key bottleneck, and we should investigate ways to lower the response time of requests to the database. Figure 8 shows that increasing the parallelism of the DB will also lower the loss rate. It should be easier to increase the parallelism of the DB than to reduce the response time. The two graphs show that the service integrator would want to negotiate to improve the guaranteed level of service in the SLA for the database service, thus increasing the number of requests that can be processed without changing the response time criterion.

The baseline results also showed that the Buy Confirm page was the dominant source of timeouts in the site. Further results (not shown) from solving the model showed that decreasing the processing requirements for the Buy Confirm page would lower the loss significantly.

6 Conclusion

In this paper we have framed the problem of coming up with SLA guarantee terms for a Web service that is composed of a collection of Web services. The problem is that of relating the SLA terms of the sub-services to the aggregate service in a useful manner. Focusing on Web-commerce, we showed how the aggregated service might need to provide types of guarantees that are different from those obtained from the sub-services. We have proposed a model of the Web services to relate the different guarantees to each other.

We realized our model using a TPC-W benchmark implementation, and set up the same implementation experimentally to relate the performance and SLA guarantees of the sub-services to the performance and SLA guarantees of the complete Web-commerce site. The results from the model agreed closely with the experimental values and also allowed us to answer questions that could not have been answered through experimentation alone. For instance, we determined that speeding up the database response time in our example would significantly improve performance. We did so by varying the database response time, which would have been very difficult to do in an actual database and demonstrates the usefulness of the model.

References

1. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (SOAP) 1.1," Tech. Rep., W3C, 2000.
2. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (WSDL) 1.1," Tech. Rep., W3C, 2001.
3. G. Clark, T. Courtney, D. Daly, D. D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, "The Möbius modeling tool," in *Proceedings of the 9th International Workshop on Petri Nets and Performance*, September 2001, pp. 241–250.
4. D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, "The Möbius framework and its implementation," *Transactions on Software Engineering*, vol. 28, no. 10, October 2002.
5. D. A. Menascé and V. A. F. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*, Prentice Hall, 2000.
6. D. A. Menascé, V. A. F. Almeida, R. Fronseca, and M. A. Mendes, "Business-oriented resource management policies for e-commerce servers," *Performance Evaluation*, vol. 42, pp. 223–239, 2000.
7. J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: Structure, behavior and application," in *Proc. International Conference on Timed Petri Nets*, 1985, pp. 106–115.
8. Transaction Processing Performance Council (TPC), *TPC Benchmark W (Web Commerce)*, August 2001.
9. Transaction Processing Performance Council (TPC) Web Page, <http://www.tpc.org>
10. *UDDI Executive White Paper*, November 2001 <http://www.uddi.org>.