RC22456 (W0205-171) May 22, 2002
Computer Science

IBM Research Report

The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services

Alexander Keller, Heiko Ludwig
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services

Alexander Keller [*]   Heiko Ludwig [†]

Abstract

We describe a novel framework for specifying and monitoring Service Level Agreements (SLA) for Web Services. SLA monitoring and enforcement become increasingly important in a Web Service environment where enterprises rely on services that may be subscribed dynamically and on demand. For economic and practical reasons, we want an automated provisioning process for both the service itself as well as the SLA management system. It measures and monitors the QoS parameters, checks the agreed-upon service levels, and reports violations to the authorized parties involved in the SLA management process.

The *Web Service Level Agreement (WSLA)* framework, our approach to these issues, is targeted at defining and monitoring SLAs for Web Services. Although WSLA has been designed for a Web Services environment, it is applicable as well to any inter-domain management scenario such as business process and service management or the management of networks, systems and applications in general. The WSLA framework consists of a flexible and extensible language based on XML Schema and a runtime architecture comprising several SLA monitoring services, which may be outsourced to third parties to ensure a maximum of objectivity. WSLA enables service customers and providers to unambiguously define a wide variety of SLAs, specify the SLA parameters and the way how they are measured, and relate them to managed resource instrumentations. Upon receipt of an SLA specification, the WSLA monitoring services are automatically configured to enforce the SLA. An implementation of the WSLA framework, the *SLA Compliance Monitor*, is publicly available as part of the IBM Web Services Toolkit.

Keywords

Service Level Agreements; Web Services; WSLA; Electronic Contracts; Service Management

1 Introduction

Emerging standards for the description, advertisement and invocation of online services promise that organizations can integrate their systems in a seamless manner. The Web Services framework [15] provides such an integration platform, based on the *Web Services Description Language (WSDL)* [32], the *Universal Discovery, Description and Integration (UDDI)* service registry [29] and, for example, the *Simple Object Access Protocol (SOAP)* as a communication mechanism. Web Services provide the opportunity to dynamically bind to services at runtime, i.e., to enter and dismiss a business relationship with a service provider on a case-by-case basis and on-demand [13]. Electronic contracts specify the way how these interactions are carried out and which contractual parties are involved. An important aspect of a contract for IT services is the set of Quality of Service (QoS) guarantees and the obligations of the various parties. This is commonly referred to as a Service Level Agreement (SLA) [30, 16].

Today, SLAs between organizations are used in all areas of IT services – in many cases for hosting and communication services but also for help desks and problem resolution. Furthermore, the parameters for which service level objectives (SLO) are defined come from a variety of areas, such as business process management, service and application management, and traditional systems and network management. In

[*] IBM Research Division, T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA, E-Mail: *alexk@us.ibm.com*

[†] IBM Research Division, T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA, E-Mail: *hludwig@us.ibm.com*

addition, different organizations have different definitions for crucial IT parameters such as Availability, Throughput, Downtime, Bandwidth, Response Time, etc. Today's SLAs are often plain natural language documents. Consequently, they must be manually provisioned and monitored, which is very expensive and slow. The definition, negotiation, deployment, monitoring and enforcement of SLAs must become - in contrast to today's state of the art - an automated process.

One approach to deal with this problem (e.g., for simple Web hosting services for consumers) is the use of SLA templates [23] that include several automatically processed fields in an otherwise natural language-written SLA. However, the flexibility of this approach is limited and only suitable for a small set of variants of the same type of service using the same QoS parameters and a service offering that is not likely to undergo changes over time. In situations where service providers must address different SLA requirements of their customers, they need a flexible formal language to express service level agreements and a runtime architecture comprising a set of services being able to interpret this language. The objective of this paper is to present our approach to such a flexible SLA specification and monitoring framework, with a focus on Web Services. It is called **Web Service Level Agreement (WSLA)** framework.

The paper is structured as follows: In section 2, we describe the underlying principles of our work. Then, we analyze the requirements of dynamic e-Businesses, both on the WSLA runtime architecture comprising multiple SLA monitoring services, and on a flexible, formal SLA language. We also describe the relationships of our work to the existing state of the art. The WSLA runtime architecture (described in section 3) provides mechanisms for accessing resource metrics of managed systems and for defining, monitoring and evaluating SLA parameters according to a WSLA specification. Section 4 introduces the WSLA language by means of several examples. It is based on XML Schema and allows parties to define QoS guarantees for electronic services and the processes for monitoring them. Section 5 concludes the paper and gives an overview of our current work.

2 Principles of the WSLA Framework

Service level management has been the subject of intense research for several years and has reached a certain degree of maturity. However, despite initial work in the field (see e.g., [2]), the problem of establishing a generic framework for service level management in cross-organizational environments remains unsolved. In section 2.1, we introduce the terminology and describe the fundamental principles, which will be used throughout this paper. Section 2.2 describes several SLA establishment scenarios. In section 2.5, we derive the requirements on the WSLA runtime architecture and language and provide an overview of related work.

2.1 Terminology

As depicted in figure 1, management information relating to SLAs appears at various tiers of a distributed system and can be classified as follows:

- **Resource Metrics** are retrieved directly from the managed resources residing in the service provider's tier, such as routers, servers and instrumented applications. Typical examples of resource metrics are the MIB variables of the IETF Structure of Management Information (SMI) [31], such as counters and gauges. To integrate resource seamlessly into a Web Services environment, WSLA uses the concept of a **Measurement Directive**. For every resource metric appearing in an SLA, a Measurement Directive is specified, which contains the command and other context information needed to retrieve the metric from the managed resource instrumentation.

- **Composite Metrics** are created by combining several resource (or other composite) metrics according to a specific algorithm, such as averaging one or more metrics over a specific amount of time, or by breaking them down according to specific criteria (top 10%, minimum, maximum values of a time series). This is usually done within the service provider's domain but can be outsourced to a third-party measurement service as well (cf. section 2.5.3). We assume that composite metrics are either specified in the SLA by means of a **Function** (a formula describing the input metrics and

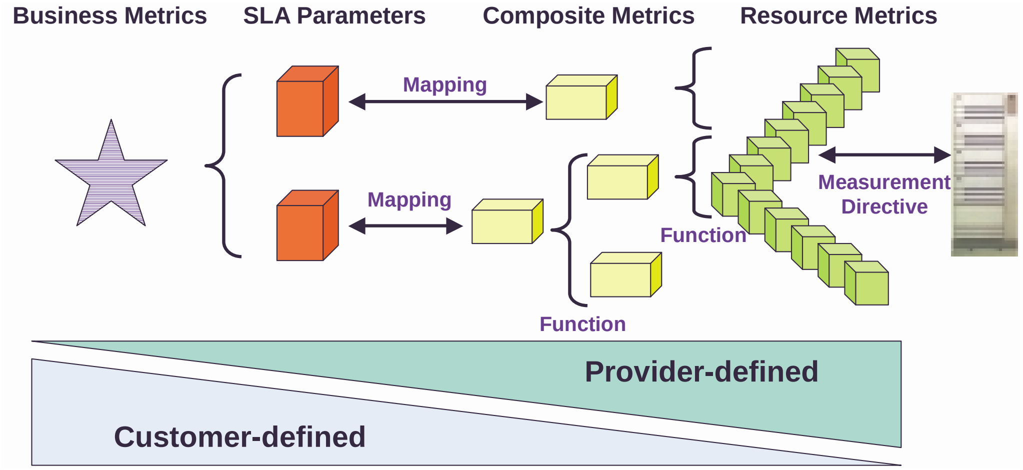

Figure 1: Aggregating Business Metrics, SLA Parameters and Metrics across different Organizations

the arithmetic operations to aggregate them) or exposed by a service provider by means of a well-defined (usually HTTP or SOAP based) interface for further processing. Note that only metrics can be aggregated into other metrics; aggregation is not defined for SLA Parameters.

- **SLA Parameters** put the metrics available from a service provider into the context of a specific customer and are therefore the core part of an SLA. In contrast to the previous metrics, every SLA parameter may be associated with high/low watermarks, which enables the customer, provider, or a designated third party to evaluate the retrieved metrics whether they meet/exceed/fall below defined service level objectives. Consequently, every SLA Parameter and its permitted range are defined in the SLA, in addition to its mapping to a metric. It makes sense to delegate the evaluation of SLA parameters against the SLOs as well to an independent third party; this ensures that the evaluation is objective and accurate.

- **Business Metrics** relate SLA parameters to financial terms specific to a service customer (and thus are usually kept confidential by him). They form the basis of a customer's risk management strategy and exist only within the service customer's domain. It should be noted that a service provider needs to perform a similar mapping to make sure the SLAs he is willing to satisfy are in accordance with his business goals.

The WSLA framework presented in this paper is designed to handle all four different parameter types; apart from the latter, they relate directly to technical management and are our main focus. However, the flexible mechanism for composing SLAs (described in detail in section 4) can be easily extended to accommodate business metrics as well. Finally, it should be noted that the responsibility for defining the four different parameter types gradually shifts from the provider to the customer (as depicted in figure 1): While a provider is primarily responsible for exposing a set of resource or composite metrics, a customer often needs to refine these according to his needs by specifying additional composite metrics. A customer is always involved - together with a provider - in the definition of SLA Parameters and needs to define his own business metrics to make sure the SLA data can be mapped to his business goals (cf. section 2.3.1 for a more detailed discussion). The next section will provide more background on the definition process by presenting the various possibilities how SLAs are established.

2.2 SLA Establishment Scenarios

Often, it is not obvious to draw a line between the aforementioned parameter types, in particular between Composite Metrics and SLA Parameters. Therefore, we assume that every parameter related to a customer and associated with a guaranteed value range is considered an SLA parameter, which is supposed to be part of an SLA. However, this distinction is also highly dependent on the extent a customer requires the customization of metrics exposed by the service provider (or a third-party measurement service) – and how much he is willing to pay for it. This, in turn, depends on the degree of customization the provider is willing to apply to the metrics he exposes. The following scenarios describe the various ways how SLAs may be defined:

1. **A customer adopts the data exposed by a service provider without further refinement.**
   This is often done when the metrics reflect good common practice, cannot be modified by the customer or are of small(er) importance to him. In this case, the selected metrics become the SLA parameters and thus integral parts of the SLA. Examples are: *length of maintenance intervals* or *backup frequency*.

2. **The customer requests that collected data is put into a meaningful context.**
   A customer is probably not interested in the overall availability of a provider's data center, but needs to know the availability of the specific cluster within the data center on which his applications and data are hosted. A provider's data collection algorithm therefore needs - at least - to take into account for which customer the data is actually collected. A provider may decide to offer such preprocessed data, such as: *Availability of the server cluster hosting customer X's web application*.

3. **The customer requests customized data that is collected according to his specific requirements.**
   While a solution to item 2 can still be reasonably static (changes tend to happen rarely and the nature of the modifiable parameters can be anticipated reasonably well), the degree of choice for the customer can be taken a step further by allowing him to specify arbitrary parameters, e.g., the input parameters of a data collection algorithm. This implies that a service provider needs to have a mechanism in place that allows a customer to provide these input parameters – preferably at runtime. E.g.: *The average load of a server hosting the customer's website should be sampled every 30 seconds and collected over 24 hours*. Note that a change of these parameters results in a change of the terms and conditions of an SLA: For example, when a customer chooses sampling intervals that impact the performance of the monitored system, which may entail the violation of SLAs the service provider has with other customers.

4. **The customer specifies the way how data is collected.**
   This means that the customer defines, in addition to the metrics and input parameters, the data collection algorithm. Obviously, this is the most extreme case and seems fairly unlikely. However, large customers may insist of getting access to very specific data that is not part of the standard set: For example, a customer may want to know which employees of a service provider had physical access to the systems hosting his data and would like to receive a daily log of the badge reader. This means that, in addition to the aforementioned extensions, a service provider needs to have a mechanism in place that allows him to introduce new data collection mechanisms without interrupting his management and production systems.

While the last case poses the highest challenge on the programmability of the monitoring system, a service provider benefits greatly from a management system being capable of handling such flexible SLAs because all the former situations are special cases of the latter. It also addresses the extreme variability of today's SLAs. Sample SLAs we analyzed (cf. section 2.5.1) clearly indicate the need for defining a mechanism that allows to unambiguously specify the data collection algorithm. Also, it should be noted that the different possibilities of specifying service level objectives are not mutually exclusive and may all be specified within the same SLA.

2.3 SLA-driven System Administration

Having introduced the concepts of SLA management, we can derive its implications on systems administration and management. While the high dynamics of the establishment and dismissal of business relationships and the resulting allocation and deallocation of system resources to different users alone is a challenge on its own, we have found several other issues that are likely to impact how system administration is done in such an environment. The way we see the tasks of a system administrator evolve are described in the following subsections.

2.3.1 Express System Resources in Financial Terms

While system administrators usually have an awareness of the costs of the systems they manage, the need to assign prices to the various resources on a very fine-grained basis will certainly increase. For quite some time, it has been common practice in well-run multi-customer data centers to account for CPU time, memory usage and disk space usage on a per-user basis. What will become increasingly important in SLA-driven system administration is the monitoring, accounting and billing of aggregated QoS parameters such as response time, throughput and bandwidth, which need to be collected across a variety of different systems that are involved in a multi-tiered server environment. Having such a fine-grained accounting scheme in place is the prerequisite for defining SLOs along with associated penalties or bonuses. In addition, the business impact of an outage or delay on the customer needs to be assessed. While the latter is mainly relevant to a service customer, a system administrator on the service provider side will need an even better understanding of the cost/benefit model behind the services offered to a customer. As a sidenote, the ability to offer measurement facilities for fine-grained service parameters is likely to become a distinguishing factor among service providers.

2.3.2 Involvement in SLA Negotiation

The technical expertise of a system administrator is likely to play an increasing role in an area that is currently confined to business managers and lawyers: The negotiation of SLAs terms. While current SLAs (see section 2.4 for more details on typical SLAs in use today) are dominated by legal terms and conditions, it will become necessary in an environment where resources are shared among different customers (under a variety of SLAs) to evaluate whether enough spare capacity is available to accomodate an additional SLA that asks for a specific amount of resources without running into the risk that the resources become overallocated if a customer's demand increases. While complex resource allocation schemes will probably not be deployed in the near future, an administrator nevertheless needs to have an understanding of the safety margins he must take into account when accepting new customers.

A related problem is to evaluate whether additional load due to SLA measurements is acceptable: While it may well be the case that enough capacity is available to accomodate the workload resulting from the service usage, overly aggressive SLA measurement algorithms may have a detrimental impact on the overall workload a system can handle. An extreme example for this is a customer whose application resides on a shared server and who would like to have the availability of the system being probed every few seconds. In this case, an SLA may either need to be rejected due to the additional workload, or the price for carrying out the measurements will need to be adjusted accordingly.

2.3.3 Classify Customers according to Revenue

The previous discussions make it clear that a service provider's approach to SLA-driven management entails the definition of enterprise policies that classify customers, e.g., according to the profit margins or their degree of contribution to a service provider's overall revenue stream. The involvement of system administrators in the process of policy definition and enforcement is a consequence of having both a high degree of technical understanding and insight into the business: First, this expertise is needed to determine which policies are reasonable and enforceable. Second, once the policies are defined, it is up to the administrator to enforce them: For example, if the resource capacity becomes insufficient because of increased workload of a high-paying customer, lower-paying customers may be starved out if the penalties associated with their SLAs can be offset by the increased gains from providing additional capacity to a higher-paying customer. Third, it should be noted that such a behavior adds an interesting twist to the problem determination schemes an administrator uses: The non-functioning of a customer's system may not necessarily be due to a technical failure, but may well be the consequence of a business decision.

2.3.4 Fix Outages according to Classification

The establishment of policies and the classification of customers also has implications on how system outages are addressed. Traditionally, system administrators are trained to address the most severe outages first. This may change if a customer classification scheme is in place, because then the system whose downtime or decreased level of service is the most expensive for the service provider will need to be fixed first. Outages are likely going to be classified not according to their technical severity, but rather based on their business impact.

2.4 Lessons learned from real-life SLAs

A suitable SLA framework for Web Services must not constrain the parties in the way they formulate their clauses but instead allow for a high degree of flexibility. A management tool that implements only a non-modifiable textbook definition of, e.g., an SLA parameter "availability" would not be considered helpful by today's service providers and their customers.

Our studies of close to three dozen SLAs currently used throughout the industry in the areas of application service provisioning (ASP) [1], web hosting and information technology (IT) outsourcing have revealed that even if seemingly identical SLA parameters are being defined, their semantics vary greatly.

While some service providers confine their definition of "application availability" to the network level of the hosting system ("user(s) being able to establish a TCP connection to the appropriate server"), others refer to the application that implements the service ("Customer's ability to access the software application on the server"). Still others rely on the results obtained from monitoring tools ("the application is accessible if the server is responding to HTTP requests issued by a specific monitoring software"), while another approach uses elaborate formulas consisting of various metrics, which are sampled over fixed time intervals.

These base clauses are then usually annotated with exceptions, such as maintenance intervals, weekend/holiday schedules, or even the business impact of an outage ("An outage has been detected by the ASP but no material, detrimental impact on the customer has occurred as a result“). The latter example, in particular, illustrates the disconnect between the people involved in the negotiation and establishment of an SLA (usually business managers and lawyers) and the ones who are supposed to enforce it (system administrators). One way of closing this gap is to enable system administrators to become involved in the negotiation of an SLA (as mentioned in section 2.3.2) by providing them with a tool able to create a legal document, namely the SLA.

It is important to keep in mind that, while the nature of the clauses may differ considerably among different SLAs, the general structure of all the different SLAs remains the same: Every analyzed SLA contains

- the involved parties,
- the SLA parameters,
- the metrics used as input to compute the SLA parameters,
- the algorithms for computing the SLA parameters,
- the service level objectives and the appropriate actions to be taken if a violation of these SLOs has been detected.

This implies that there is a way to come up with a SLA language that can be applied to a multitude of bilateral customer/provider relationships. Our approach to such a language is presented in section 4.

2.5 WSLA Design Goals

In this section, we will derive – based on the above discussions – the requirements the WSLA framework needs to address.

2.5.1 Flexible, formal Language to accomodate a wide Variety of SLAs

In the introduction of this paper, we have stressed the point that SLAs, their parameters and the SLOs defined for them are extremely diverse. One approach to deal with this problem (e.g., as it is done today for simple consumer Web hosting services) is to narrow down the "universe of discourse" to a few well-understood terms and to limit the possibilities of choosing arbitrary QoS parameters through the use of SLA templates [23]. SLA templates include several automatically processed fields in an otherwise natural language-written SLA. However, the flexibility of this approach is limited and only suitable for a small set of variants of the same type of service using the same QoS parameters and a service offering that is not likely to undergo changes over time. In situations where service providers must address different SLA requirements of their customers, they need a more flexible formal language to express service level agreements and a runtime architecture comprising a set of services being able to interpret this language. The WSLA runtime architecture is detailed in section 3; the WSLA language is described in section 4.

2.5.2 Integration with Electronic Commerce Systems

Architectural components and language elements related to SLA negotiation, creation and deployment should be compatible with existing approaches and systems developed in the electronic commerce and B2B area. This applies in particular to the advertisement, negotiation, and sales of SLA-based services. Electronic storefronts that handle basic order processing and payment are available from many major software companies, e.g., IBM's WebSphere Commerce Suite and mySAP. In addition, electronic marketplaces such as Ariba's or CommerceOne's are in widespread use for manufacturing materials and supplies and could be extended to services. Sophisticated matchmaking technology such as IBM's WebSphere Matchmaking Edition [6] can be applied to finding suitable offerings for products with many complex features as in SLAs. Bichler [4] provides an overview of current marketplace technology. Since SLA based services can be quite unique, providers and their customers may want to negotiate their SLAs individually, e.g., by defining specific metrics for a customer. Automated negotiations and negotiation middleware are the subject of current research, e.g., in the context of the SilkRoad [26] and SeCo [10] projects. The notion of agreeing on contracts and deploying them has been a subject of research in the past years – particularly for connecting business processes across organizations. There are description languages for B2B interaction, e.g., in the ebXML stack [5]. Other work deals with contracts for monitoring and managing outsourced processes, e.g., CrossFlow [9]. A number of approaches deals with electronic contracts and their deployment in general [18]. A summary of electronic contracting-related projects can be found in [9].

2.5.3 Delegation of Monitoring Tasks to third Parties

Traditionally, an SLA is a bilateral agreement between a service customer and a service provider: The *enhanced Telecom Operations Map (eTOM)* [27], for example, defines various roles services providers can play. Additional work in this area has been carried out within the scope of the IST Project FORM [7], which addresses SLAs in an inter-domain environment. FORM also deals with the important issue of federated accounting [3], which we do not address in this paper. However, the current state of the art does not provide flexible mechanisms for the delegation of management functionality from a service provider and customer to further (third party) service providers. We refer to the parties that establish and sign the SLA as **signatory parties**.

SLA monitoring may require the involvement of third parties: They come into play when either a function needs to be carried out that neither service provider nor customer wants to do, or if one signatory party does not trust the other to perform a function correctly. Third parties act then in a **supporting role** and are sponsored by either one or both signatory parties. Figure 2 gives an overview of a configuration where two signatory parties and two supporting parties collaborate in the monitoring of an SLA.

Service provider (`ACMEProvider` in figure 2) and service customer (`XInc`) are the signatory parties to the SLA. They are ultimately responsible for all obligations, mainly in the case of the service provider, and (in the case of the customer) the ultimate beneficiary of obligations. Supporting parties are sponsored either by one or both of the signatory parties to perform one or more of a particular set of roles. A measurement service (`YMeasurement`) implements a part or all of the measurement and computation activities defined within an SLA. A condition evaluation service (`ZAuditing`) implements violation detection and other state checking functionality that covers all or a part of the guarantees of an SLA. A management service implements corrective actions.

Note that these services (described in more detail in section 3.1) are distributed among the various parties and need to interact across organizational domains. There can be multiple supporting parties having a similar role, e.g., a measurement service may be located in the provider's domain while another measurement service probes the service offered by the provider across the Internet from various locations. Keynote Systems, Inc. [14] is a real-life example of such an external measurement service provider. SLA monitoring issues in multi-provider environments are described in [22] and [21].

2.5.4 Deploying SLAs: The "Need to know" Principle

As motivated in the previous section 2.5.3, the functionality of computing SLA parameters or evaluating contract obligations may be split, e.g., among multiple measurement or SLO evaluation services, each provided by a different organization. On the other hand, all the definitions and obligations of the involved signatory and supporting parties should be defined within *a single* SLA document, which fully describes the contractual relationships. Hence, it is important that every supporting service receives only the parts of an SLA it needs to know to carry out its task: a service dealing with the deployment of an SLA document to the various involved parties needs to verify the obligations of every party and distribute only the relevant parts to them. Since SLAs with multiple involved parties may become fairly complex, this is not a trivial task. Section 3.1.2 presents our approach for dealing with this problem.

Since it may be possible that a signatory party delegates the same task (e.g., response time probing)

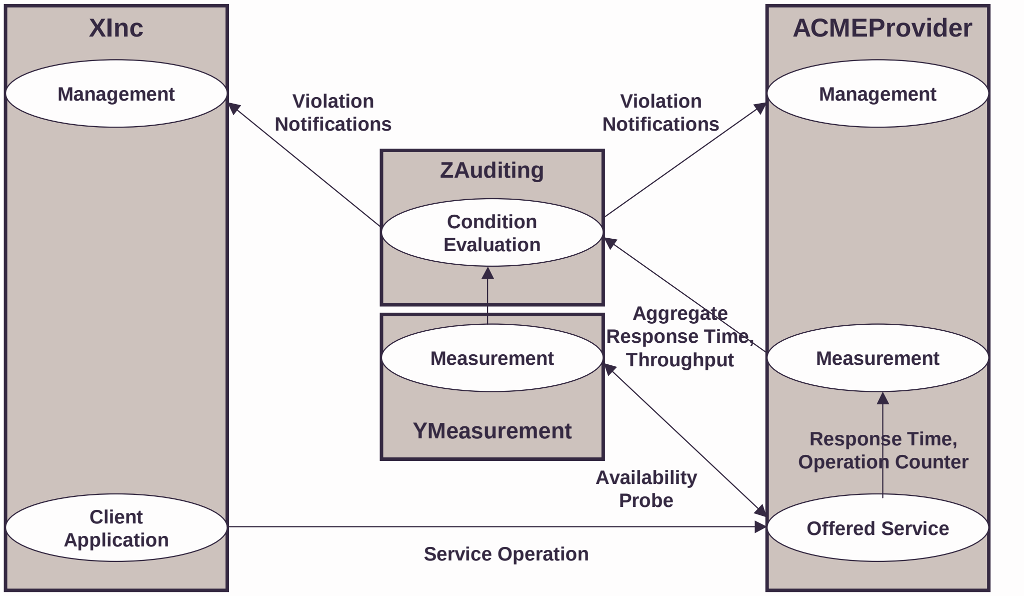

Figure 2: SLA Management with Multiple Service Providers

to several different supporting parties (in order to be able to cross-check their results), different service instances may not be aware of other instances. Stated differently, signatory parties specify in the SLA from where a supporting party retrieves its input data and where to send its results. Consequently, a supporting service becomes aware of the existence of other (supporting) services only if the signatory parties have stated this in the part of the SLA he receives.

Another major issue that underlines the importance of the "Need to know" principle are the privacy concerns of the various parties involved in an inter-domain management scenario: A service provider is, in general, neither interested in disclosing which of his business processes have been outsourced to other providers, nor the names of these providers. On the other hand, service customers will not necessarily see a need to know the exact reason of performance degradations as long as a service provider is able to take appropriate remedies (or compensate its customer for the incurred service level violation).

Traditionally, end-to-end performance management has been the goal of traditional enterprise management efforts and is often explicitly listed as a requirement (see, e.g., [24]). However, the aforementioned privacy concerns of service providers and the service customers' need for transparency make that an end-to-end view becomes unachievable (and irrelevant) in an e-Business on demand environment spanning multiple organizational domains.

2.5.5 SLA-driven Configuration of Managed Resources

Since the terms and conditions of an SLA may entail setting configuration parameters on a potentially wide range of managed resources, an SLA management framework must accommodate the definition of SLAs that go beyond electronic/web services and relate to the supporting infrastructure. On the one hand, it needs to tie the SLA to the monitoring parameters exposed by the managed resources so that an SLA monitoring infrastructure is able to retrieve important metrics from the resources. [31] defines a MIB for SLA performance monitoring in an SNMP environment, whereas the SLA handbook from TeleManagement Forum [25] proposes guidelines for defining SLAs that target telecom service providers. The capability of mapping resource metrics to SLA parameters is crucial because a service provider must be able to answer the following questions before signing an SLA:

- Is it possible to accept an SLA for a specific service class given the fact that the capacity is limited?
- Can additional workload be accomodated?

On the other hand, it is desirable to derive configuration settings directly from SLAs. However, the heterogeneity and complexity of the management infrastructure makes configuration management a challenge; section 3.1.4 discusses this problem. Successful work in this area often focuses on the network level: [8] describes a network configuration language; the Policy Core Information Model (PCIM) of the IETF [20] provides a generic framework for defining policies to facilitate configuration management. Existing work in the e-commerce area may be applied here as well since the concept of contract-driven configuration in e-commerce environments [11] and virtual enterprises [18, 12] has similarities to the SLA-driven configuration of managed resources.

3 WSLA Runtime Architecture

In this section, we break down the WSLA framework into its atomic building blocks, namely the elementary services needed to enable the management of an SLA throughout the stages of its lifecycle. The first part, section 3.1, describes the information flows and interactions between the different WSLA services. Section 3.2 describes our prototype implementation.

3.1 Interactions between the WSLA Services

The services described in this section are designed to address the "need to know" principle (motivated in section 2.5.4) and constitute the atomic building blocks of our SLA monitoring framework. The WSLA

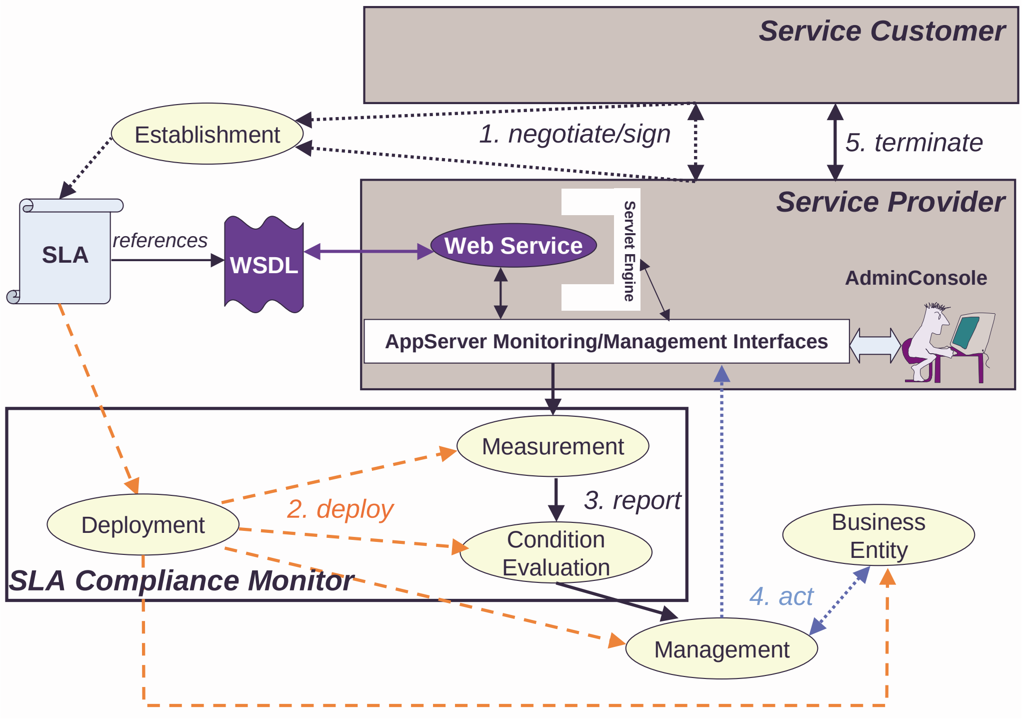

Figure 3: WSLA Services and their interactions

services are intended to interact across multiple domains; however, it is possible that some services may be co-located within a single domain and not necessarily exposed to the ones residing within another domain.

Figure 3 gives an overview of the SLA management lifecycle, which consists of five distinct stages. We assume that an SLA is defined for a web service, which is running in the servlet engine of a web application server. The web application server exposes a variety of management information either through the graphical user interface of an administration console or at its monitoring and management interfaces, which are accessed by the various services of our SLA monitoring framework. The interface of the web service is defined by an XML document in the *Web Services Description Language (WSDL)*. The SLA references this WSDL document and extends the service definition with SLA management information. Typically, an SLA defines several SLA parameters, each referring to an operation of the web service. However, an SLA may also reference the service as a whole, or even compositions of multiple web services [28]. The stages and the services that implement the functionality needed during the various stages are as follows:

3.1.1 Stage 1: SLA Negotiation and Establishment

The SLA is being negotiated and signed by both signatory parties. This is done by means of an **SLA Establishment Service**, i.e., an SLA authoring tool that lets both signatory party establish, price and sign a SLA for a given service offering. This tool allows a customer to retrieve the metrics offered by a service provider, aggregate and combine them into various SLA parameters, request approval from both parties, define secondary parties and their tasks, and make the SLA document available for deployment to the involved parties (dotted arrows in Figure 3). Note that, as stated in section 2.5.4, the outcome of the negotiation process is a single SLA document comprising the relationships and obligations of all the involved signatory and supporting parties.

3.1.2 Stage 2: SLA Deployment

Deployment Service: The deployment service is responsible for checking the validity of the SLA and distributing it either in full or in appropriate parts to the involved components (dashed arrows in Figure 3). Since two signatory parties negotiate the SLA, they must inform the supporting parties about their respective roles and duties. Two issues must be addressed:

1. Signatory parties do not want to share the whole SLA with their supporting parties but restrict the information to the relevant information such that they can configure their components. Further, signatory parties must analyze the SLA and extract relevant information for each party. In the case of a measurement service (described in the next section 3.1.3), this is primarily the definition of SLA parameters and metrics. SLO evaluation services obtain the SLOs they need to verify. All parties need to know the definitions of the interfaces they must expose, as well as the interfaces of the partners they interact with.

2. Components of different parties cannot be assumed to be configurable in the same way, i.e., they may have heterogeneous configuration interfaces.

Thus, the deployment process contains two steps. In the first step, the SLA deployment system of a signatory party generates and sends configuration information in the *Service Deployment Information (SDI)* format (omitted for the sake of brevity), a subset of the language described in section 4, to its supporting parties. In the second step, deployment systems of supporting parties configure their own implementations in a suitable way.

3.1.3 Stage 3: Service Level Measurement and Reporting

This stage deals with configuring the runtime system in order to meet one or a set of SLAs, and with carrying out the computation of SLA parameters by retrieving resource metrics from the managed resources and executing the management functions (solid arrows in figure 3). The following services implement the functionality needed during this stage:

Measurement Service: The Measurement Service maintains information on the current system configuration, as well as run-time information on the metrics that are part of the SLA. It measures SLA parameters such as availability or response time either from inside, by retrieving resource metrics directly from managed resources, or outside the service provider's domain, e.g., by probing or intercepting client invocations. A Measurement Service may measure all or a subset of the SLA parameters. Multiple Measurement Services may simultaneously measure the same metrics. The elements of the WSLA language relating to the tasks of a Measurement Service are described in section 4.1.

Condition Evaluation Service: This service is responsible for comparing measured SLA parameters against the thresholds defined in the SLA and notifying the management system. It obtains measured values of SLA parameters from the Measurement Service and tests them against the guarantees given in the SLA. This can be done each time a new value is available, or periodically. Section 4.2 describes the language elements a Condition Evaluation Service needs to understand.

3.1.4 Stage 4: Corrective Management Actions

Once the Condition Evaluation Service has determined that an SLO has been violated, corrective management actions need to be carried out. The functionality that needs to be provided in this stage spans two different services:

Management Service: Upon receipt of a notification, the Management Service (usually implemented as part of a traditional management platform) will retrieve the appropriate actions to correct the problem, as specified in the SLA. Before acting upon the managed system, it consults the Business Entity (see below) to verify if the proposed actions are allowable. After receiving approval, it applies the action(s) to the managed system. It should be noted that the Management Service seeks approval for every proposed action from the Business Entity (dotted arrows in the lower right part of figure 3). The main purpose of the

Management Service is to execute corrective actions on behalf of the managed environment if a Condition Evaluation Service discovers that a term of an SLA has been violated. While such corrective actions are limited today to opening a trouble ticket or sending an event to the provider's management system, we envision this service playing a crucial role in the future by acting as an automated mediator between the customer and provider, according to the terms of the SLA. This includes the submission of proposals to the management system of a service provider on how a performance problem could be resolved (e.g., proposing to assign a different traffic category to a customer if several categories have been defined in the SLA). Our implementation addresses very simple corrective actions; finding a generic, flexible and automatically executable mechanism for corrective management actions remains an open issue yet, because there is no standard for submitting corrective actions to a management platform.

Business Entity: This conceptual component represents the embodiment of business knowledge, goals and policies of a signatory party (here: service provider), which are usually not exposed to the business partner. It is involved in decision-making on management operations proposed by the Management Service. The Business Entity either approves the proposal of the Management Service or derives another management operation based on its knowledge of the state of the system and the specific business-related information it has access to. Business-related information can come from many sources: A Customer Relationship Management (CRM) system may indicate that a good customer is affected, whose requests must be prioritized although the load the customer is putting on the system is higher than specified in the SLA. The accounting system – implemented, e.g., using SAP R3 or another Enterprise Resource Planning (ERP) system – may indicate that a customer exceeded his credit line with the service provider, assuming that the service is pay-per-use, thus rejecting any further request from this customer. In case decision-making is more complex and relies on ”good judgement”, employees are part of the ”system” implementing the Business Entity. The implementation of the Business Entity will be different from organization to organization. Due to its complexity we did not implement a prototype Business Entity that can be connected to various sources of business information.

We have experienced that the tasks covered by these two services become extremely complicated as soon as sophisticated management actions need to be specified: First, a service provider would need to expose what management operations he is able to execute, which is very specific to the management platforms (products, architectures, protocols) he uses. Second, these management actions may become very complicated and may require human interaction (such as deploying new servers). Finally, due to the fact that the provider's managed resources are shared among various customers, management actions that satisfy an SLA with one customer are likely to impact the SLAs the provider has with other customers. The decision whether to satisfy the SLA (or deliberately break it) therefore is not a technical decision anymore, but rather a matter of the provider's business policies and, thus, lies beyond the scope of the work discussed in this paper. Consequently, only few elements of the WSLA language (cf. section 4) address this stage of the service lifecycle.

3.1.5 Stage 5: SLA Termination

The SLA may specify the conditions under which it may be terminated or the penalties a party will incur by breaking one or more SLA clauses. Negotiations for terminating an SLA may be carried out between the parties in the same way as the SLA establishment is being done. Alternatively, an expiration date may be specified in the SLA.

3.2 SLA Compliance Monitor Implementation

Figure 3 also depicts which WSLA services we have implemented. The general-purpose **Measurement Service** supports metric definitions using a rich set of functions. It features multiple data providers – plug-ins that interpret and execute measurement directives to read measurement data – e.g., the metering service of the IBM Web Services Toolkit (WSTK). Other data providers can be added. Measurement Services have a Web Services interface to exchange metric values during runtime. In addition, a general-purpose **Condition Evaluation Service** has been implemented that supports a wide range of predicates. It offers a

Web Services interface to receive metric updates from Measurement Services. The **Deployment Service** decomposes WSLA documents into parts relevant for particular Measurement Services and Condition Evaluation Services. It also provides a simple WSLA repository and functions for the lifecycle management of SLAs, e.g., to deactivate the monitoring of SLAs. In addition, a WSLA Authoring Service (as a first step towards an SLA Establishment Service supporting automated negotiation) has been implemented to support the template-based creation of WSLA offering templates and the filling of those templates at subscription time.

These services are implemented as Web Services themselves and are jointly referred to as **SLA Compliance Monitor**, which acts as a wrapper for them. The SLA Compliance Monitor is included in the current version 3.2 of the IBM Web Services Toolkit and can be downloaded from http://www.alphaworks.ibm.com/tech/webservicestoolkit. Our ongoing implementation efforts, aimed at completing the WSLA framework, are described in section 5.

4 WSLA Language

The WSLA Language Specification [19] defines a type system for the various SLA artifacts. It is based on XML Schema [33, 34]. In principle, there are many variations of what types of information and which rules are to be included in a specific SLA. However, as discussed in section 2.5.1, there is a common understanding on how the general structure of an SLA looks like. WSLA is designed to accomodate this structure in three sections:

- The **Parties** section identifies all the contractual parties. **Signatory Party** descriptions contain the identification and the technical properties of the parties, i.e., their interface definition (e.g., the way they accept events) and their addresses. The definitions of the **Supporting Parties** contain, in addition to the information contained in the signatory party descriptions, an attribute indicating the sponsor(s) of the party. Since the information contained in this section is straightforward, we will not discuss the corresponding language elements in detail.

- The **Service Description** section of the SLA specifies the characteristics of the service and its observable parameters. This information is processed by a Measurement Service; the parts of the WSLA language dealing with this information are described by means of various examples in section 4.1.

- **Obligations**, the last section of an SLA, define various guarantees and constraints that may be imposed on SLA parameters. In section 4.2, we focus on these parts of the WSLA language and present two typical examples. The Condition Evaluation Service needs to understand this information to evaluate if a service level objective has been violated.

In the following two sections, we will highlight the major elements of the WSLA language by means of a comprehensive and detailed example. The example assumes a multi-party environment (as depicted in figure 2) in which a Service Provider `ACMEProvider`, a Measurement Service `YMeasurement` and a Condition Evaluation Service `ZAuditing` cooperate to enact an SLA.

4.1 Service Description: Defining the SLA Parameters of a Service

The purpose of the service description is the clarification of four issues: *What are the SLA parameters? To which service do they relate? How are SLA parameters measured or computed? How are the Metrics of a managed resource accessed?* This is the information a Measurement Service requires to carry out its tasks. A sample service description is depicted in Figure 4. For the operation `getQuote` of a Web Service, two SLA parameters `AvgThroughput` (average transaction throughput) and `OverUtilization` (percentage of time the service provider's system experiences a workload that is above the agreed-upon threshold) are defined.

The rationale for choosing these two parameters is as follows: SLAs are defined under the assumption that the ranges of SLA parameters defined for a service reflect typical workloads. In practice, a service

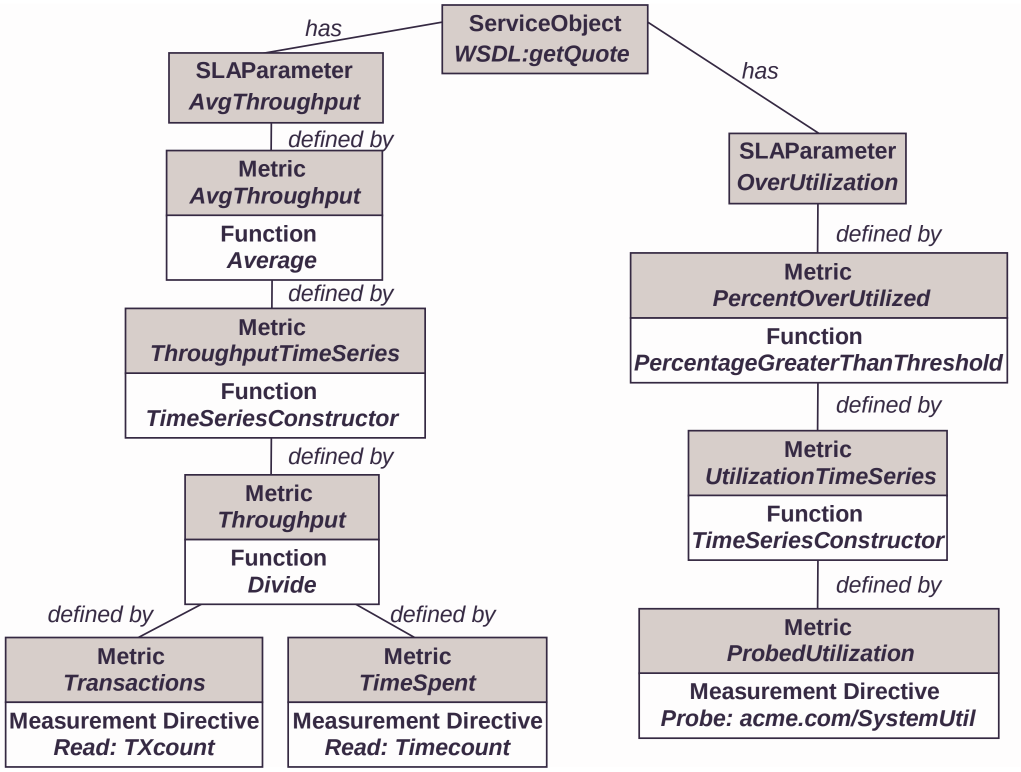

Figure 4: Sample elements of a service description

provider has authority over some environmental factors while others are beyond his control. Thus, an SLA needs to take into account under which conditions the obligations are valid. Assigning simply a threshold to an SLA parameter is not helpful without considering the variations of workload to which a service provider's system may be exposed, because sudden load surges may increase the workload on the system by several multiples. An increase of the workload by, e.g., a factor of 5 or more makes it impossible for a service provider to meet fixed response time or throughput targets. Thus, in our example, `OverUtilization` will serve in section 4.2 as a precondition to constrain under which circumstances the service provider needs to guarantee a given `AvgThroughput`.

The various parts relating to the definition of the various WSLA elements for specifying the way how the measurements are carried out will be discussed subsequently. For the sake of brevity, our descriptions will detail the definitions of how the SLA parameter `OverUtilization` is computed.

4.1.1 Service Objects and Operations

The service object, depicted at the top of Figure 4, provides an abstraction of all conceptual elements for which SLA parameters and the corresponding metrics can be defined. In the context of Web Services, the most detailed concept whose quality aspect can be described separately is an individual **Service Operation** described in a WSDL specification [32]. For every Service Operation, one or more **Bindings**, i.e., the transport encoding for the messages to be exchanged, may be specified. Examples of such bindings are SOAP (Simple Object Access Protocol) over HTTP (HyperText Transfer Protocol) or MIME (Multipurpose Internet Mail Extensions). In our example, the operation `getQuote` is the service object, which may contain references to operations in a WSDL file. Outside the scope of Web Services, business processes, or parts thereof, can be service objects (e.g., defined in WSFL [17]).

```
<SLAParameter name="OverUtilization" type="float" unit="Percentage">
   <Metric>PercentOverUtilized</Metric>
   <Communication>
     <Source>YMeasurement</Source>
     <Pull>ZAuditing</Pull>
     <Push>ZAuditing</Push>
   </Communication>
</SLAParameter>
```

Figure 5: Defining an SLA Parameter `OverUtilization`

4.1.2 SLA Parameters and Metrics

SLA Parameters are properties of a service object; each SLA parameter has a name, type and unit. Examples of SLA parameters are *service availability*, *throughput*, or *response time*. As mentioned in section 2.1, every SLA parameter refers to one (composite) **Metric**, which, in turn, aggregates one or more other (composite or resource) metrics. This aggregation can be done in two ways: a metric either defines a **Function** that can use other metrics as operands or it has a **Measurement Directive** (see below) that describes how the metric's value should be measured, i.e., how it can be retrieved from a managed resource. Examples of composite metrics are *maximum response time of a service*, *average availability of a service*, or *minimum throughput of a service*. Examples of resource metrics are: *system uptime*, *service outage period*, *number of service invocations*.

Since SLA parameters are surfaced by a Measurement Service to a Condition Evaluation Service, it is important to define which party is supposed to provide the value (`Source`) and which parties can receive it, either event-driven (`Push`) or through polling (`Pull`). Note that one of our design choices is to restrict the aggregation mechanism to Metrics only, i.e., no SLA parameters can be defined as input parameters for computing other SLA parameters. In Figure 4, one metric is retrieved by probing a web based interface (`acme.com/SystemUtil`) while the other ones (`TXcount`, `Timecount`) are directly retrieved from the service provider's management system. In our example, YMeasurement retrieves the Metric `ProbedUtilization` from ACMEProvider.

Figure 5 depicts how an SLA parameter `OverUtilization` is defined. It is assigned the metric `PercentOverUtilized`, which is defined independently of the SLA parameter for being used potentially multiple times. YMeasurement promises to send (`Push`) new values to ZAuditing, which is also allowed to retrieve new values on its own initiative (`Pull`).

A **Function** represents a measurement algorithm (or formula) that specifies how a composite metric is computed. Examples of functions are formulas of arbitrary length containing mean, median, sum, minimum, maximum, and various other arithmetic operators, or time series constructors.

Figure 6 depicts two sample composite metrics having the datatypes float and TS, a WSLA type to represent time series. `YMeasurement` is in charge of computing the values of both metrics. `UtilizationTimeSeries` is of type TS and has no unit. The example illustrates the concept of a function: Every 5 minutes, a new value of the metric `ProbedUtilization` is placed by the function of type `TSConstructor` into a time series for further processing.

The second Metric `PercentOverUtilized` is used to determine the amount of time when a system is overloaded and expresses this as a percentage. In our example, we consider a system utilization of less than 80% as a safe operating region; above this value, the system is considered overloaded. Specific functions, such as *Minus*, *Mean*, *Median* or, here, *PercentageGreaterThanThreshold* (yielding the percentage of values over a threshold in a time series, in our example 0.8 or 80%) are extensions of the common function type. Operands of functions can be metrics, scalars and other functions. It is expected that a Measurement Service, provided either by a signatory or a supporting party, is able to compute functions. More specific and customized functions can be added to the WSLA language as needed.

Every function references either a **Schedule** or a **Trigger**. A schedule defines the time intervals during which the functions are executed to compute the metrics. These time intervals are specified by means

```
<Metric name="PercentOverUtilized" type="float" unit="Percentage">
    <Source>YMeasurement</Source>
    <Function xsi:type="PercentageGreaterThanThreshold" resultType="float">
        <Schedule>BusinessDay</Schedule>           <!-- defined separately -->
        <Metric>UtilizationTimeSeries</Metric>
        <Value>
           <LongScalar>0.8</LongScalar>            <!-- 80% -->
        </Value>
    </Function>
</Metric>

<Metric name="UtilizationTimeSeries" type="TS" unit="">
    <Source>YMeasurement</Source>
    <Function xsi:type="TSConstructor" resultType="float">
        <Schedule>Every5Minutes</Schedule>        <!-- defined separately -->
        <Metric>ProbedUtilization</Metric>
        <Window>12</Window>
    </Function>
</Metric>
```

Figure 6: Defining a Metric `PercentOverUtilized`

of *Start*, *End*, and *Interval*. Examples of the latter are *weekly*, *daily*, *hourly*, or *every minute*. Arbitrary combinations are possible. Note that we have omitted the schedule definitions in our example for the sake of brevity. Alternatively, a trigger defines a point in time to which the execution of monitoring activity can be tied. In figure 6, the first function has a reference to a schedule `BusinessDay`, which specifies when and how often the data is supposed to be collected during working days. Since we assume for our example that this schedule provides the collection of metrics on an hourly basis, we need to make sure that enough new values are present in the time series at any point in time. We achieve this by setting the `Window` size of a time series to 12, because a new measurement is placed in the time series every 5 minutes. In our implementation, time series are implemented as ring buffers with a user-defined window size, thus making it easy to compute moving averages or to accomodate different measurement intervals or clock drift on the involved systems. Also note that different functions may reference different schedules, thus enabling the definition of highly customizable measurements.

A **Measurement Directive**, depicted in figure 7, specifies *how* an individual metric is retrieved from the source (either by means of a well-defined query interface offered by the service provider, or directly from the instrumentation of a managed resource by means of a management protocol operation). Typical examples of measurement directives are the uniform resource identifier of a hosted computer program, a protocol message, or the command for invoking scripts or compiled programs.

In the above example, a specific type of measurement directive `Gauge` is used to retrieve the current value of the metric `ProbedUtilization` (depicted in the lower right corner of figure 4). It contains a URL that is used for probing the value of the `SystemUtil` gauge. Apparently, other ways to measure values require an entirely different set of information items, e.g., an SNMP port, an object identifier (OID) and an instance identifier to retrieve a counter.

```
<Metric name="ProbedUtilization" type="float" unit="">
   <Source>ACMEProvider</Source>
   <MeasurementDirective xsi:type="Gauge" resultType="float">
      <RequestURL>http://acme.com/SystemUtil</RequestURL>
   </MeasurementDirective>
</Metric>
```

Figure 7: Defining a Measurement Directive for the Metric `ProbedUtilization`

4.2 Obligations: SLOs and Action Guarantees

Obligations, the last section of an SLA, define various guarantees and constraints that may be imposed on the SLA parameters. This allows the parties to unambiguously define the respective guarantees they give each other. The WSLA language provides two types of obligations:

- **Service Level Objectives** represent promises with respect to the state of SLA parameters.
- **Action Guarantees** are promises of a signatory party to perform an action. This may include notifications of service level objective violations or invocation of management operations.

Important for both types of obligations is the definition of the obliged party and the definition of when the obligations need to be evaluated. Both have a similar syntactical structure; however, their semantics are different. The content of an obligation is refined in a service level objective (see section 4.2.1 below) or an action guarantee (described in section 4.2.2).

4.2.1 Service Level Objectives

A service level objective expresses a commitment to maintain a particular state of the service in a given period. Any party can take the obliged part of this guarantee; however, this is typically the service provider. In WSLA, an SLO has the following elements: `Obliged` is the name of a party that is in charge of delivering what is promised in this guarantee. One or more `ValidityPeriod`s define when the SLO is applicable. Examples of validity periods are *business days*, *regular working hours* or *maintenance periods*.

A logical `Expression` defines the actual content of the guarantee, i.e., what is asserted by the service provider to the service customer. Expressions follow first order logic and contain the usual operators *and*, *or*, *not*, etc., which connect either predicates or, again, expressions. Predicates (*greater than*, *equal*, *less than*, etc.) are used to specify thresholds against which SLA parameters are compared. Consequently, they can have SLA parameters or scalar values as parameters. The result of a predicate is either *true* or *false*. By extending an abstract predicate type, new domain-specific predicates can be introduced as

```
<ServiceLevelObjective name="Conditional_SLO_For_AvgThroughput">
   <Obliged>ACMEProvider</Obliged>
   <Validity>
     <Start>2001-11-30T14:00:00.000-05:00</Start>
     <End>2001-12-31T14:00:00.000-05:00</End>
   </Validity>
   <Expression>
     <Implies>
       <Expression>
         <Predicate xsi:type="Less">
           <SLAParameter>OverUtilization</SLAParameter>
           <Value>0.3</Value>                  <!-- 30% -->
         </Predicate>
       </Expression>
       <Expression>
         <Predicate xsi:type="Greater">
           <SLAParameter>AvgThroughput</SLAParameter>
           <Value>1000</Value>
         </Predicate>
       </Expression>
     </Implies>
   </Expression>
   <EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
```

Figure 8: Defining a Service Level Objective `Conditional_SLO_For_AvgThroughput`

needed. Similarly, expressions may be extended e.g., to contain variables and quantifiers. This provides the expressiveness to define complex states of the service.

A service level objective may also have an `EvaluationEvent`, which defines when the expression of the service level objective should be evaluated. The most common evaluation event is `NewValue`, i.e., each time a new value for an SLA parameter used in a predicate is available. Alternatively, the expression may be evaluated according to a `Schedule`. A schedule is a sequence of regularly occurring events. It can be defined either within a guarantee or may refer to a commonly used schedule (cf. the discussion in section 4.1.2).

The example in figure 8 illustrates a service level objective given by ACMEProvider and valid for a full month in the year 2001. It guarantees that the SLA parameter `AvgThroughput` must be greater than 1000 if the SLA parameter `OverUtilization` is less than 0.3, i.e., the service provider must make sure his system is able to handle at least 1000 transactions per second under the condition that his system is operating under normal load conditions for 70% of the time. If the service provider experiences an overload condition for 30% of the time (due, e.g., to an excessive amount of incoming requests), he is not obliged to fulfill the `AvgThroughput` requirement. Note that in our example, overload is defined as a system utilization of at least 80% for a period of one hour (see the definition of the metric `PercentOverUtilized` in section 4.1.2). This condition should be evaluated each time a new value for the SLA parameter is available. The example shows how the `Implies` element can be used for defining preconditions in WSLA.

Note that we deliberately chose that validity periods are always specified with respect to a single SLO, and thus are only indirectly applicable to the scope of the overall SLA. Alternatively, validity periods to the overall SLA (possibly in addition to the validity periods for each SLA parameter) could be possible, but we found this granularity too coarse.

4.2.2 Action Guarantees

An action guarantee expresses a commitment to perform a particular activity if a given precondition is met. Any party can be the obliged of this kind of guarantee. This particularly includes also the supporting parties of the SLA.

An action guarantee comprises the following elements and attributes: `Obliged` is the name of a party that must perform an action as defined in this guarantee. A logic `Expression` defines the precondition of the action. The format of this expression is the same as the format of an expression in service level objectives. An important predicate for action guarantees is the `Violation` predicate that determines whether another guarantee, in particular a service level objective, has been violated. An `EvaluationEvent` or an evaluation `Schedule` defines when the precondition is evaluated.

`QualifiedAction` contains a definition of the action to be invoked at a particular party. The concept of a qualified action definition is similar to the invocation of an object's method in a programming language, replacing the object name with a party name. The party of the qualified action can be the obliged or another party. The action must be defined in the corresponding party specification. In addition, the specification of the action includes the marshalling of its parameters. One or more qualified actions can be part of an action guarantee. Examples of qualified actions are: *sending an event to one or more signatory and supporting parties*, *opening a trouble ticket or problem report*, *payment of penalty*, or *payment of premium*. Note that, as stated in the latter case, a service provider may very well receive additional compensation from a customer for exceeding an obligation.

`ExecutionModality` is an additional means to control the execution of the action. It can be defined whether the action should be executed if a particular evaluation of the expression yields true. The purpose is to reduce, for example, the execution of a notification action to a necessary level if the associated expression is evaluated very frequently. Execution modality can be either: *always*, *on entering a condition* or *on entering and leaving a condition*. The example depicted in figure 9 illustrates an action guarantee.

In the example, `ZAuditing` is obliged to invoke the notification action of the service customer `XInc` if a violation of the service level objective `Conditional_SLO_For_AvgThroughput` (cf. Figure 8) occurs. The precondition should be evaluated every time the evaluation of the SLO `Must_Send_Notification_Guarantee` returns a new value. The action has three parameters: the type of

```
<ActionGuarantee name="Must_Send_Notification_Guarantee">
  <Obliged>ZAuditing</Obliged>
  <Expression>
    <Predicate xsi:type="Violation">
      <ServiceLevelObjective>Conditional_SLO_For_AvgThroughput</ServiceLevelObjective>
    </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
  <QualifiedAction>
    <Party>XInc</Party>
    <Action actionName="notification" xsi:type="Notification">
      <NotificationType>Violation</NotificationType>
      <CausingGuarantee>Must_Send_Notification_Guarantee</CausingGuarantee>
      <SLAParameter>AvgThroughput OverUtilization</SLAParameter>
    </Action>
  </QualifiedAction>
  <ExecutionModality>Always</ExecutionModality>
</ActionGuarantee>
```

Figure 9: Defining an ActionGuarantee `Must_Send_Notification_Guarantee`

notification, the guarantee that caused it to be sent, and the SLA parameters relevant for understanding the reason of the notification. The notification should always be executed.

5 Conclusions, Current Status and Outlook

This paper has introduced the novel WSLA framework for specifying and monitoring SLAs for Web Services. Our work is motivated by the need to enable service customers and providers to unambiguously define a wide variety of SLAs, specify the SLA parameters and the way how they are measured, and tie them to managed resource instrumentations. Upon receipt of an SLA specification, the SLA monitoring services are automatically configured to enforce the SLA, thus reducing the need for costly, slow and error-prone manual intervention to a minimum. This becomes increasingly important for emerging service-oriented architectures, such as Web Services.

The WSLA framework addresses these problems by allowing service providers and their customers to define the quality of service aspects of a service, and Web Services in particular. In order to avoid the potential ambiguity of higher-level SLA parameters, parties can define precisely how resource metrics are measured and how composite metrics are computed. The concept of supporting parties allows signatory parties to include third parties into the process of measuring the SLA parameters and monitoring the obligations associated with them. The WSLA language is extensible and allows to derive new domain-specific or technology-specific elements from existing language elements. The explicit representation of service level objectives and action guarantees provides a very flexible mechanism to define obligations on a case-by-case basis. Finally, its independence from the way how the interface of a service is described makes the WSLA language and its associated services applicable to a wide range of inter-domain management scenarios.

We have developed a prototype of a WSLA Compliance Monitor. It consists of a measurement service, a condition evaluation service, and a deployment service. This prototype is publicly available on the IBM Alphaworks site as part of the IBM Web Services Toolkit (www.alphaworks.ibm.com). Currently, we provide extensions to the WSLA language that apply to quality aspects of business processes and pricing. The integration with existing resource management systems and architectures remains a challenging topic for further research.

Acknowledgments

The authors express their gratitude to Asit Dan, Richard Franck, Richard P. King, Robert E. Moore and Lee M. Rafalow for their contribution.

References

[1] ASP Industry Consortium. *White Paper on Service Level Agreements*, 2000.

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

[3] B. Bhushan, M. Tschichholz, E. Leray, and W. Donnelly. Federated Accounting: Service Charging and Billing in a Business-To-Business Environment. In N. Anerousis, G. Pavlou, and A. Liotta, editors, *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, pages 107–121, Seattle, WA, USA, May 2001. IEEE Publishing.

[4] M. Bichler. *The Future of e-Markets - Multidimentional Market Mechanisms*. Cambridge University Press, Cambridge, United Kingdom, 2001.

[5] *ebXML – Creating a Single Global Electronic Market*. http://www.ebxml.org.

[6] S. Field, C. Facciorusso, Y. Hoffner, A. Schade, and M. Stolze. Design Criteria for a Virtual Marketplace (ViMP). In C. Nikolaou and C. Stephandis, editors, *Research and Advanced Technology for Digital Libraries*, Berlin, 1998. Springer-Verlag.

[7] FORM Consortium. Final Inter-Enterprise Management System Model. Deliverable 11, IST Project FORM: Engineering a Co-operative Inter-Enterprise Framework Supporting Dynamic Federated Organisations Management, February 2002. http://www.ist-form.org.

[8] R. Gopal. Unifying Network Configuration and Service Assurance with a Service Modeling Language. In R. Stadler and M. Ulema, editors, *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, pages 711–725, Florence, Italy, April 2002. IEEE Publishing.

[9] P.J. Grefen, K. Aberer, H. Ludwig, and Y. Hoffner. Crossflow: Cross-organizational workflow management for service outsourcing in dynamic virtual enterprises. *IEEE Data Engineering Bulletin*, 24(1):52–57, 2001.

[10] M. Greunz, B. Schopp, and K. Stanoevska-Slabeva. Supporting Market Transactions through XML Contracting Container. In *Proceeding of the 6th Americas Conference on Information Systems (AMCIS 2000)*, Long Beach, CA, 2000.

[11] F. Griffel, M. Boger, H. Weinreich, W. Lamersdorf, and M. Merz. Electronic contracting with COSMOS - How to establish, Negotiate and Execute Electronic Contracts on the Internet. In *Proceedings of the Second International Enterprise Distributed Object Computing Workshop (EDOC '98)*, La Jolla, CA, USA, October 1998.

[12] Y. Hoffner, S. Field, P. Grefen, and H. Ludwig. Contract-driven Creation and Operation of Virtual Enterprises. *Computer Networks*, 37:111–136, 2001.

[13] A. Keller, G. Kar, H. Ludwig, A. Dan, and J.L. Hellerstein. Managing Dynamic Services: A Contract based Approach to a Conceptual Architecture. In R. Stadler and M. Ulema, editors, *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, pages 513–528, Florence, Italy, April 2002. IEEE Publishing.

[14] *Keynote – The Internet Performance Authority*. http://www.keynote.com.

[15] H. Kreger. *Web Services Conceptual Architecture 1.0*. IBM Software Group, May 2001.

[16] L. Lewis. *Managing Business and Service Networks*. Kluwer Academic Publishers, 2001.

[17] F. Leymann. *Web Services Flow Language (WSFL) 1.0*. IBM Software Group, May 2001.

[18] H. Ludwig and Y. Hoffner. The Role of Contract and Component Semantics in Dynamic E-Contract Enactment Configuration. In *Proceedings of the 9th IFIP Workshop on Data Semantics (DS9)*, pages 26–40, Hong Kong, 2001.

[19] H. Ludwig, A. Keller, A. Dan, R. Franck, and R.P. King. *Web Service Level Agreement (WSLA) Language Specification*. IBM Corporation, July 2002.

[20] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy Core Information Model - Version 1 Specification. RFC 3060, IETF, February 2001.

[21] C. Overton. On the Theory and Practice of Internet SLAs. *Journal of Computer Resource Measurement*, 106:32–45, April 2002. Computer Measurement Group.

[22] C. Overton and E. Siegel. Experiences with Internet Measurements and Statistics. *Journal of Computer Resource Measurement*, 106:4–14, April 2002. Computer Measurement Group.

[23] G. Dreo Rodosek and L. Lewis. Dynamic Service Provisioning: A User–Centric Approach. In O. Festor and A. Pras, editors, *Proceedings of the 12th Annual IFIP/IEEE International Workshop on Distributed Systems: Operations & Management (DSOM 2001)*, pages 37–48, Nancy, France, October 2001. IFIP/IEEE, INRIA Press.

[24] SLA and QoS Management Team. Service Provider to Customer Performance Reporting: Information Agreement. Member Draft Version 1.5 TMF 602, TeleManagement Forum, June 1999.

[25] SLA Management Team. SLA Management Handbook. Public Evaluation Version 1.5 GB 917, TeleManagement Forum, June 2001.

[26] M. Ströbel. *A Design and Implementation Framework for Multi-Attribute Negotiation Intermediation in Electronic Markets*. PhD thesis, Universität St. Gallen, St. Gallen, Switzerland, 2002.

[27] enhanced Telecom Operations Map: The Business Process Framework. Member Evaluation Version 2.7 GB 921, TeleManagement Forum, April 2002.

[28] V. Tosic, B. Pagurek, B. Esfandiari, and K. Patel. Management of Compositions of E- and M-Business Web Services with multiple Classes of Service. In R. Stadler and M. Ulema, editors, *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002)*, pages 935–937, Florence, Italy, April 2002. IEEE Publishing.

[29] UDDI Version 2.0 API Specification. Universal Description, Discovery and Integration, uddi.org, June 2001.

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

[31] K. White. Definition of Managed Objects for Service Level Agreements Performance Monitoring. RFC 2758, IETF, February 2000.

[32] Web Services Description Language (WSDL) Version 1.2. W3C Working Draft, W3 Consortium, July 2002.

[33] XML Schema Part 1: Structures. W3C Recommendation, W3 Consortium, May 2001.

[34] XML Schema Part 2: Datatypes. W3C Recommendation, W3 Consortium, May 2001.

Biography

Alexander Keller is a Research Staff Member at the IBM Thomas J. Watson Research Center in Yorktown Heights, NY, USA. He received his M.Sc. and a Ph.D. in Computer Science from Technische Universität München, Germany, in 1994 and 1998, respectively and has published more than 30 refereed papers in the area of distributed systems management. He does research on service and application management, information modeling for e-business systems, and service level agreements. He is a member of the USENIX Association, the IEEE and the DMTF CIM Applications Working Group.

Heiko Ludwig is a Visiting Scientist at the IBM Thomas J. Watson Research Center since June 2001. As a member of the Distributed Systems and Services department he works in the field of electronic contracts, both contract representation and architectures for contract-based systems. He holds a Master's degree (1992) and a Ph.D. (1997) in computer science and business administration from Otto-Friedrich University Bamberg, Germany.