This paper describes a framework for providing differentiated levels of Web services to different customers on the basis of service level agreements (SLAs). Service providers can offer Web services at different service levels -- for example, at bronze, silver, and gold levels with increasingly better response times. In general, the service levels are differentiated based on many variables such as responsiveness, availability, and performance. The framework described in this paper comprises the Web Service Level Agreement (WSLA) language to specify SLAs in a flexible and individualized way, a system to monitor the compliance of a provided service with a service level agreement, and a workload management system that prioritizes requests according to the associated SLAs. Using this framework, service providers can efficiently and flexibly manage their resources to optimize customer satisfaction and, potentially, yield.

Overview

A service level is used to define the expected performance behavior of a deployed Web service, where the performance metrics are, for example, average response time, supported throughput, service availability, etc. During deployment of a Web service, the resources of an underlying Web service container can be reconfigured (by acquiring new resources, if necessary) to provide a certain service level. Even the same Web service can be offered at different service levels to different clients by dynamically allocating resources for execution of individual Web service requests. Hence, to receive assurances on the service level, a client creates a priori a service level agreement (SLA) associated with this Web service with the service provider.

In many different business scenarios, successful deployment and use of Web services are critically dependent on guaranteeing and managing the service levels of deployed Web services. With the emerging models of dynamic e-business, i.e., eUtility, eSourcing, eHosting and eBusiness-on-demand, where an application service provider (ASP) provides an outsourced service to many clients, the same service may be offered at different service levels for a different price [eBOD, WSTK02, LDKK02, SAM02]. In any case, the customers expect a certain performance level as guaranteed by the service provider through individualized SLAs. Violation of any pre-agreed service levels results in a penalty assessment on the part of the service provider. Similarly, a
service application may be hosted by an infrastructure resource provider guaranteeing a service level for this hosted service using an SLA. This is similar to providing different airline reservation services with platinum, gold, and other classes of services.

Service level specification and management are equally important considerations for the deployment of Web services within an enterprise.

- First, in a large organization, SLAs are used across divisions to provide service level assurances [ITSLA].
- Second, service level management is simply a good design practice for an application server to protect the infrastructure resources from a runaway workload (e.g., due to a faulty application or high traffic generated by a malicious user).
- Third, it provides an easy mechanism for a system administrator to intelligently provision and monitor system resources [Horn02, Autonomic].
- Finally, this is also the fundamental vision in many of the key IBM initiatives, namely Autonomic computing [Horn02, Autonomic], Grid computing [Grid] and eBusiness-on-Demand [eBOD] -- a vision of self-configuring and self-managing system that dynamically acquires/allocates resources based on a business level service level goals.

In a business-to-business interaction such as procurement and supply chain, where the primary objective is not application outsourcing but business collaboration, an SLA between the partners plays an equally important role in the overall business agreement. Without assurances on the availability of the partner sites, supported transaction rate, and response time, a business may lose its capability to conduct normal operation. Also note that in B2B collaboration, SLAs may include not just IT-level assurances, but also guarantees on the business responsiveness, e.g., delivery of goods within a certain time duration. Therefore, until businesses step up to supporting service levels associated with B2B applications via appropriate management of their resources, B2B applications will not flourish as anticipated widely.

Formal specification of an SLA is therefore a key element of the Web service specification stack. In this paper we describe how an SLA can be specified, as well as the key components of a service hosting architecture for managing SLAs: creation, deployment, enforcement, and monitoring. The key issues and requirements in SLA specification, i.e., flexibility in specification of service level objectives, unambiguity in specification -- in order to enable external monitoring of service levels by customers and possibly even by a third party, etc. -- are specified in the section below entitled "SLA compliance monitoring." It also includes an overview of a Web service level agreement (WSLA) specification language first supported in the IBM Web services toolkit (WSTK 3.2) released as alphaWorks code [WSTK02]. The WSLA specification language will certainly evolve to keep consistent with other developments in the Web services specification stack and to assimilate feedback from our initial experience. However, many of the core concepts will remain intact.

Figure 1 illustrates the key components of an architecture for differentiating services. At the top level, the component marked "Subscription and high-level SLA management" is responsible for all external customer interactions regarding SLA: creation of service offerings and associated service levels, creation of an SLA via customer subscription and any further customization via negotiation, billing, and service level report generation, notification of SLA termination,
renegotiation, etc. At the next level, a customer SLA is used for provisioning appropriate resources and for compliance monitoring. While not on the direct path of every service access, these components are responsible for initial deployment and resource provisioning based on an SLA, as well as dynamic resource provisioning based on monitored and predicated data. It also monitors SLA compliance based on measured data and detects any violation to be used by the customer interaction component. Finally, at the lowest level, the workload management component enforces SLA by controlling admission, prioritizing requests, etc. Certain functions of this level are invoked on every request, while other functions adjust weights and algorithm parameters asynchronously. The workload management component receives information about deployed SLAs and provisioned resources from the provisioning component, and in turn provides measurement data to the compliance monitor component for analysis.

**Figure 1. Overview of an architecture for differentiating services**

The remainder of the paper is organized as follows. In the section entitled "SLA creation and life-cycle management," we first provide an overview of managing an SLA life cycle, i.e., creation, deployment and provisioning, enforcement and compliance monitoring, and termination. (This section provides an overview of provisioning as part of the SLA life-cycle management. However, further details on provisioning are beyond the scope of the current paper. Generally speaking, details on resource provisioning depend very much on the specific resource management environment. However, some abstractions on dynamic resource acquisition can be found in the paper for Grid computing [Topol03].) Next, in the section entitled "SLA compliance monitoring," we discuss key issues and requirements in SLA specification, especially in making it unambiguous for compliance monitoring. We provide an architecture for distributed compliance monitoring that can involve external third-parties. In the next section, entitled "WSLA concepts and syntax," we provide an overview of the WSLA specification, and discuss how this unambiguous specification is used in compliance monitoring. The next section, "Workload management," specifies a conceptual architecture for workload management, i.e., how incoming requests are classified as per SLA specified goals, and the control mechanisms are used in managing these goals. We summarize our key points at the conclusion of the article.
SLA creation and life-cycle management

An SLA represents a joint agreement, and hence is created with the input from both the parties, i.e., service customer and provider. However, depending on the business scenario, the process may be quite asymmetric. Under an eUtility model, a service provider typically creates a set of service offerings -- fixed packages including service operations, associated service levels, penalty upon violation, as well as price for using this service -- expressed as SLA templates. A customer subscribes to a selected service offering creating a new SLA. The service provider may provide some customization flexibility in its offerings. The customization capability may range from mere selection of a few SLA parameter values (e.g., from a set of fixed throughput levels) as expressed in an offer, to some negotiation of parameters (e.g., negotiation of price for a customer-specified throughput level) to composing new service level objectives. Of course, to provide this flexibility, a provider should not only have the required capability of online negotiation, but also its business ability to support any new customer-specified service level objectives (SLOs), i.e., runtime infrastructure for supporting this service level as well as its ability to price this new service level. Therefore, before accepting a new SLA, the provider must ensure its ability to support this new SLA.

The life cycle of an SLA may be broadly classified into four phases: creation, deployment and provisioning, enforcement and monitoring for service invocation under an SLA, and, finally, SLA termination. Depending on the business scenario, there may be many subphases within each phase. Also, some provisioning activity may take place prior to creation of an SLA, and/or deferred until runtime invocation of a service.

SLA creation

This phase culminates in the creation of a joint agreement between a service provider and a customer. As illustrated earlier, the creation process may be as simple as a customer selecting from a set of pre-specified offers, or extensive customization via a negotiation process. In either case, the provider needs to create a service offering based on its known infrastructure capability and knowledge of service details, e.g., service operations. After a Web service is deployed to a hosting application server (i.e., its application code is registered with the application server), a business administrator may create multiple service offerings with different associated service levels, price, and/or penalty using an offer creation tool [WSTK02]. The offer is represented as an SLA template where in addition to a partially completed SLA document, the template specifies modifiable and/or negotiable fields as well as any constraints to be followed by the customer during customization.

In the scenario where the offer is kept flexible for customization via online negotiation, the hosting environment is also set up for supporting this negotiation. During online negotiation, a provider specified business logic module may be invoked for computing a price associated with a new service level, as well as to check if such a service level can be supported by this provider.

In the context of a Grid service [Topol03], the service offerings and SLA templates can be associated with a Grid service factory. Here, prior to creation of a new service instance, an SLA is created either by selection of template or via customization. The SLA defines the service level provided by the new grid service instance.
SLA deployment and provisioning

After an SLA is created with a customer, the deployment and provisioning step involves two distinct issues:

1. Provisioning of a service-providing system according to the service level objectives defined in the SLA. This is an issue for the service provider only.
2. Setup of the SLA monitoring and management environment. This issue may be relevant for both service provider and service customer.

The service provider configures the application server for enabling access by this customer, provisions sufficient infrastructure resources to support the service level guaranteed in the SLA, and notifies the workload management layer of this new SLA. This resource provisioning may involve translating high-level objectives specified in an SLA to a detailed configuration of a hosting environment [CV02]. Determining optimal configuration requires detailed understanding of the resource and service performance characteristics, which is not always easy to obtain. Therefore, to facilitate this translation, pre-defined configurations may be used as building blocks, and SLA objectives are used to select closest (multiple of) pre-defined configurations and/or extrapolate to certain parameterization of this pre-defined configuration.

Exploiting economies of scale, a provider always takes a certain amount of risk by allocating fewer dedicated resources than is necessary for supporting peak throughput for a customer. The service level will be violated if the peak throughput is demanded by all customers simultaneously, which is a rare occurrence [Kienzle96]. In any case, in a more adaptive environment, dynamic provisioning of resources based on near-violation and/or predicted workload can address this dynamic fluctuation in resource requirement [CD02].

The deployment process also includes setting up a distributed monitoring environment, i.e., data aggregation and checking service level objectives as detailed in the section entitled "SLA compliance monitoring."

In the context of a Grid service [Topol03], a service instance is provisioned to meet the SLA requirements. The provisioning process includes setting up a monitoring environment for monitoring actual service levels provided by an instance.

SLA enforcement and monitoring

During runtime invocation of a service, a provider monitors the service level as per the associated SLA with this customer and actively manages resources to avoid any violation of guarantees. This includes prioritization of requests to be served based on service level assessment [WSTK02] and/or dynamic allocation of resources by assigning a thread priority [eWLM]. The provider also controls customer access to a service so that it doesn't exceed the guaranteed throughput level. This is detailed further in the section entitled "Workload management."

As described earlier, a customer may also monitor externally the service level received to avoid any blind trust on a provider. In some scenarios, the two parties may agree to use a third
party for monitoring this service level. Obviously, this is possible if the third party is able to measure independently the service level either via special probe transactions, or by receiving raw performance data from multiple sources (e.g., customer and provider; see the section entitled "SLA compliance monitoring").

Any violation of guarantees are noted for future penalty assessment and/or dynamically notified to the parties to this agreement. Upon monitoring such violations, the customer may choose to terminate its SLA with the provider. The provider may use this violation (as well as alerts on potential future violations) to dynamically provision new resources [CD02]. When a provider is not able to meet all its commitments, it may prioritize its business commitments using various business objectives (e.g., profit maximization, preferential treatment of loyal customers, etc.) [SAM02], and in the worst scenarios terminate certain SLAs.

**SLA termination**

An SLA specifies a validity period, after which the deployed SLA is terminated. The SLA may also be terminated explicitly either by the customer or the provider (due to the change in requirements of a customer and/or capability of the provider). The business and legal implications of such a termination is outside the scope of this document. The termination may also be initiated due to multiple/excessive violations of guarantees specified in an SLA. Finally, an SLA may be renegotiated to extend the validity period and/or to agree on a new service level and price.

**SLA compliance monitoring**

**Issues and requirements**

The SLA specification defines the agreed level of performance for a particular service as shared between a service provider and service customer. Having analyzed a set of current SLAs, we found that many SLAs contains a similar set of key elements: the involved parties, the SLA parameters, the base metrics used as input to compute the SLA parameters, the algorithm for computing the SLA parameters, the service guarantees, and the appropriate actions to be taken if a violation of these guarantees has been detected. Here are the requirements on scope and expressiveness of the SLA specification:

- In the Web services context, service level guarantees and the relevant input parameters must be associable with individual Web services, as defined in a WSDL file, or processes of Web services, as defined in a BPEL4WS specification. Service level guarantees must be definable for individual operations in bindings. Definitions on the port type level are only of limited use. For example, response times can only be expressed meaningfully per operation since the processing of different operations in the application service takes a different amount of time independent of the current load. Different bindings also may yield different response times for the same operation.
- Potentially, a large diversity of SLA parameters can be attached to an electronic service, depending on different needs of service customers. Even seemingly simple parameters such as response time can be defined in many different ways: measured from client, application server or application; individual or averaged; averaging interval; average per operation type
of a service or for each operation individually. The SLA language must provide a means to describe how SLA parameters are measured and aggregated from resource metrics.

• The SLA specification must be able to express a large variety of contractual obligations. This includes service level objectives with respect to SLA parameters as well as guaranteed courses of actions if violations occur or other critical situations arise.

• Third parties must be integrated in the supervision procedures of the SLA, primarily for making measurements that neither party can or wants to perform (e.g., probing a service from outside the provider's domain). Also, third parties are important if the signatory parties do not trust each other's measurements or condition evaluations.

• The SLA must be represented formally for automatic processing. This is important for the automatic provisioning of a service and the automatic setup of the service level monitoring and management infrastructure.

• The functionality of computing SLA parameters, evaluating contract obligations and guarantees may be split among multiple organizations, e.g., the provider, the customer, and an independent party such as Keynote. It is important that each third party receives only the part of the contract it needs to know to carry out its task. We need a mechanism to send the relevant contract fragments to the different sponsored parties.

This extensive set of requirements is addressed by the WSLA language [WSTK02, LDK02].

Model of SLA monitoring

For describing the distributed monitoring capabilities that arise from the requirements above, the SLA specification language requires a general model of the entities involved in retrieving resource metrics, aggregating metrics to useful SLA parameters, evaluating service level objectives, and taking appropriate management actions. The model must be sufficiently abstract and general to accommodate various different monitoring scenarios and system architectures. WSLA proposes a simple model, as illustrated in Figure 2. In the WSLA monitoring model, measurements can be taken from multiple measurement services at different places as illustrated. These basic inputs are called resource metrics. Measurement services aggregate resource metrics to higher-level SLA parameters on which the service level objectives are based. Measurement services can interact to exchange metrics, both resource and composite, that are input to a further aggregation of another measurement service. SLA parameters are sent to condition evaluation services that check the service level objectives based on those parameters. Multiple condition evaluation services can evaluate the same SLOs, e.g., one of the customer and one of the providers. The condition evaluation services create notifications of violations, compliance, or other service states that are sent to the management service of the customer or provider. The management service can then decide which course of action to take. The provider can change its system configuration to adapt to the current situation and can take accounting provision to book agreed penalties. On the customer side, too, management actions can be taken, e.g., the request rate to the service can be reduced. In most cases, however, the customer will simply request a reduced price if service level violations occur.
Figure 2. SLA monitoring model

The SLA specification must include the specification of which measurement service performs which measurement, which condition evaluation service checks which SLO, and to which management service notifications must be sent about what. The management actions taken within the customer and provider are not part of the SLA. However, predefined management actions can be triggered by violation notifications.

Setting up runtime environment for compliance monitoring

One of the primary purposes of the WSLA language is to facilitate the automatic setup of the functions needed to manage and supervise the SLA. Since WSLA supports a distributed model of monitoring a service that may include external parties, the deployment of a new WSLA to the monitoring infrastructure must be carefully orchestrated.

In general, each signatory party to the contract (i.e., provider and customer) deploys its monitoring infrastructure independently. However, each signatory party must ensure that the supporting third parties that it sponsored also receive the necessary information. If a third party is sponsored by both signatory parties, the service provider is in charge of setting up the sponsoring parties in question.

Two important issues must be addressed in this distributed, cross-organizational deployment scenario:

- **Information hiding:** Signatory parties do not want to share the whole SLA with their supporting parties, but rather restrict the information to the relevant information so that they can configure their components. Signatory parties must analyze the SLA and extract relevant information for each party. In the case of a measurement service, this is primarily the definition of SLA parameters and metrics. Condition evaluation services receive the obligations they must supervise. All parties need to know the definitions of the interfaces they must expose, as well as the interfaces of the partners they interact with.

- **Heterogeneity of configuration interfaces:** Components of different parties cannot be assumed to be configurable in the same way -- i.e., they may have heterogeneous configuration interfaces.
Therefore, only the minimal required SLA information is passed to the participating parties using a standard format. The deployment process contains two steps, illustrated in Figure 3.

**Figure 3. SLA deployment**

1. In the first step, the SLA deployment function of a signatory party generates and sends configuration information in the *SLA Deployment Information* (SDI) format for its supporting parties.

2. In the second step, SDI deployment functions of supporting parties configure their own implementations in a suitable way to play their role in the process of supervising the SLA.

In Figure 3, a signatory party sponsors two third parties, a measurement service and a condition evaluation service. Both are supplied with their individual SDIs, SDI₁ and SDI₂. An additional management service is located within the primary party and is also configured using an SDI, SDI₃ in the figure. Each measurement or condition evaluation service has a local service deployment function that interprets the SDI and sets up its particular, propriety measurement infrastructure.

The syntax of the SDI format is equivalent to the syntax of the WSLA language. Rather than containing a complete WSLA document, it only contains information that is relevant for a particular party. For example, a measurement service only needs to know how to retrieve and aggregate the metrics that it is in charge of and how to interact with other parties either to obtain other metrics or to make them available in the form of SLA parameters. Which obligations are given in the WSLA document or which parties participate that the measurement service does not interact with is irrelevant. On the other hand, condition evaluation services do not need to know how SLA parameters are defined by metrics. There is an SDI format for measurement services and for condition evaluation services.

Using the SDI version of WSLA, we have a universal means of communicating SLA deployment information for distributed monitoring systems across organizational boundaries.

**WSLA concepts and syntax**

This section describes the elements of the WSLA language, a formal language to define service level agreements. However, this is not a complete definition of the language but a discussion of
the semantics of the most important language elements and some examples of the XML-based syntax.

**Structure**

In general, a contract, and thus an SLA, defines the relationship between two or more parties. To do so:

- It must define the contracting parties and their properties relevant for the particular domain (e.g., addresses, accounts and interfaces).
- It must establish agreement on the concepts that are relevant for the subject matter of the contract (e.g., definition of the products or service, delivery modalities, and legal modalities). This is the definitions section of a traditional contract.
- Finally, based on the common understanding of the subject matter, it contains the contractual obligations (i.e., what is guaranteed by one party to other parties).

A WSLA document is structured correspondingly, taking into accounts the specifics of an SLA environment. It comprises the following parts:

- **Parties**: Describes the parties involved in the management of the Web service. This includes the signatory parties as well as third parties that are brought into the SLA to act on behalf of service provider or customer but cannot be held liable on the grounds of this SLA. The relationship of a sponsored party to their sponsor is not within the scope of this agreement.
- **Service definitions**: Describe the service properties on which obligations are defined. The task of establishing a common view -- or the ontology -- of the service is the definition of the service whose quality is defined within the SLA, the SLA parameters and the way they are measured and computed. To facilitate automatic processing, an SLA contains for each SLA parameter the metrics that are used as input parameters to compute its value. For each metric, in turn, it is described either how it is measured (e.g., by using monitoring interfaces or by probing), or how it is aggregated from lower-level metrics (e.g., by averaging over a time series of measured metrics).
- **Obligations**: Define the service level that is guaranteed with respect to the SLA parameters defined in the service definition section. Also, the promises to perform actions under particular conditions are represented in this part.

An SLA contains one **parties** section; it may contain multiple **service definitions** and one **obligations** section.

**Approach to the WSLA language**

The SLA language is based on XML and an XML schema has been defined for its syntax. All language elements are defined as XML Schema types. At the same time, these type definitions also help us defining the semantics of the SLA language as every element (or tag) must be assigned a type that defines the structure of its content. In the WSLA language we associate a particular meaning with a type of the WSLA XML schema, such as party, metric, function, etc., which is discussed in this section.
Since the proposed language should capture a wide range of service level agreements, an important issue is to be able to describe very heterogeneous artifacts that are relevant for the SLA. For example, the description of how to measure a particular value in a system depends on the kind of system, the method of measurement (e.g., probing and reading from a system’s instrumentation) and the interface that it exposes. A corresponding measurement directive must contain the relevant (platform-specific) access specification (e.g., a counter of an application server may be obtained from an SNMP management agent). This mandates that the SLA language be open to the definition of new types of elements. For this purpose, the relevant types of the SLA XML Schema can be extended to accommodate domain or technology specific description needs.

Parties

It is usually straightforward to define for each commitment who is the obliged and who is the beneficiary of the commitment. However, in a contract containing more than two parties, it is not obvious which party guarantees what to whom. A clear definition of responsibilities is required. The WSLA monitoring model foresees multiple parties to act as measurement or condition evaluation services, also to third parties. We approach the issue of responsibility by distinguishing two classes of parties:

- The service provider and service customer are the signatory parties to the contract. They are ultimately responsible for all obligations, mainly in the case of the service provider, and the ultimate beneficiary of obligations.
- Signatory parties can sponsor third parties to support the enactment of the contract, which we call supporting parties. Supporting parties are sponsored to perform one or more roles according to the monitoring model (e.g., a measurement service). Supporting parties can be sponsored by one or both of the signatory parties. In addition, there can be multiple supporting parties having a similar role. Each of the parties is defined by its unique name, its contact details, and the definition of the interfaces of actions that it offers, e.g., for receiving a notification or an updated SLA parameter value. The interface definitions are specified in WSDL and used by other parties for enactment time interaction. These interface definitions extend a common, well-known set of interfaces (that should become standards) such as notification and parameterUpdate by specific binding information. In addition, domain-specific actions can be agreed upon by the parties.

Additionally, supporting parties have a definition of their sponsors.

Service definition: Common ontology

The purpose of the service definition part of the language is the specification of the parties' common view of the service, the SLA's ontology. As outlined above, this primarily refers to the clarification of three issues:

1. To which element of a service do SLA parameters relate?
2. What are the SLA parameters that describe the relevant properties of this service element?
3. How are the SLA parameters measured or computed?
The service definition part of the contract provides language constructs to describe an SLA's ontology.

Figure 4 illustrates the main elements of WSLA's service definition part. SLA parameters are observable properties of a service object that are used to define the service level objectives of an SLA. A service object describes SLA parameters associated with a service. In the example, the service object is a reference to the WSDL-defined operation getQuote. It could also be a group of operations having the same SLA parameters or a business process. SLA parameters are defined by metrics. Metrics either define how a value is to be computed from other metrics or describe how it is measured. For this purpose, 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. The format of measurement directives strongly depends on the measurement method, and the monitoring and test interfaces of the service provider's system.

**Figure 4. Example SLA parameter definition**

In the example in Figure 4, the service object getQuote refers to a WSDL operation and has two SLA parameters, average response time and throughput, each of which defined by a metric of the respective name. The resource metrics on which the SLA parameters are based are the gauges TimeCount, the total amount of time spent to process getQuote requests, and TxCount, the total number of invocations. We assume that these gauges are available from the application server's instrumentation. This instrumentation access is described in the respective measurement directives for the resource metrics. Using the time series constructor function, time series are built over the reading of the resource metrics (schedule not in figure). To yield the average response time, the latest reading from each time series is taken and the time divided by the number of invocations. For the metric throughput, the last element of the throughput time series is selected.
Service objects and operations

The service object provides an abstraction for 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 the individual operation (in a binding) described in a WSDL specification. In our example, the operation `getQuote` is the service object. In addition, quality properties of groups of WSDL operations can be defined -- the operation group being the service object in this case. Outside the scope of Web services, business processes, or parts thereof, can be service objects (e.g., define in BPEL).

Service objects define a set of **SLA parameters**, a set of **metrics** that describe how SLA parameters are computed or measured, and a reference to the service itself that is the subject of the service object abstraction. While the format for SLA parameters and metrics is the same for all services (though not their individual content), the reference to the service depends on the particular way in which the service is described. For example, service objects may contain references to operations in a WSDL file.

SLA parameters and metrics

**SLA parameters** are defined properties of a service object. SLA parameters are assigned a metric that defines how its value is measured or computed. Each SLA parameter has name, type and unit (see Listing 1). Since SLA parameters are the entities that are surfaced by a measurement service to a condition evaluation service, it is important to define which party is supposed to provide the value and which parties can receive it.

**Listing 1. SLA parameter example**

```
<SLAParameter name="AverageResponseTime"
  type="float"
  unit="seconds">
  <Metric>AverageResponseTime</Metric>
  <Communication>
    <Service>ACMEProvider</Service>
    <Pull>XYZAuditing</Pull>
    <Push>ACustomer</Push>
  </Communication>
</SLAParameter>
```

Listing 1 shows an SLA parameter called `AverageResponseTime`. It is assigned the metric `AverageResponseTime`, which is defined independently of the SLA parameter so that it can potentially be used multiple times. `ACMEProvider` promises to send new values to `XYZAuditing` (push) and allows `ACustomer` to pull current values.

A **metric**'s purpose is to define how to measure or compute a value. Besides a name, a type, and a unit, it contains either a function or a measurement directive and a definition of the party that is in charge of computing this value. **Listing 2** shows an example composite metric containing a function.

**Listing 2. Metric example**

```
<Metric name="AverageResponseTime"
  type="double" unit="seconds">
  <Source>ACMEProvider</Source>
```

A metric's purpose is to define how to measure or compute a value. Besides a name, a type, and a unit, it contains either a function or a measurement directive and a definition of the party that is in charge of computing this value. **Listing 2** shows an example composite metric containing a function.
This example describes the metric average response time of Figure 4. The metric is of type double and its unit is seconds. ACMEProvider will measure its value. In its function definition, the function Divide is applied to two operands which in turn are again functions. The function TSSelect yields elements of a time series. The element "0" means the most recent value. Specific functions, such as Minus, Plus, or TSSelect are extensions of the common function type. Operands of functions can be metrics, scalars and other functions. It is expected that a measurement service implementation is able to compute functions. Specific functions can be added to the standard set as needed.

A measurement directive is the most heterogeneous element of the SLA language. It has no common elements but a type, only elements in specific extensions. A specific type of measurement directive is used in our example, Gauge. Its attributes are the name of the gauge and the operation to which it belongs.

Listing 3. Measurement directive example

```xml
<Metric name="TXCount" type="integer">
  <Source>YMeasurement</Source>
  <MeasurementDirective xsi:type="Gauge" resultType="integer">
    <Gauge>TXCount</Gauge>
    <OperationName>getQuote</OperationName>
  </MeasurementDirective>
</Metric>
```

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).

Obligations

Based on the common ontology established in the service definition part of the WSLA document, the parties can unambiguously define the respective guarantees that they give each other. The obligations section of the SLA may contain any number of guarantees. The SLA language provides two kinds of guarantees:
• **Service level objectives** represent promises with respect to the state of SLA parameters.
• **Action guarantees** are promises to perform an action. This may include notifications of service level objective violations or invocation of management operations. Important for both types of guarantees is the definition of the obliged party and definition of when they need to be evaluated. The actual definition of the guarantees' content is specific to each type.

**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. A service level objective has the following elements:

- The *obliged* is the name of a party that is in charge of delivering what is promised in this guarantee.
- One or many *validity periods* define when the guarantee is applicable.
- A logic *expression* defines the actual content of the guarantee, i.e., what is asserted by the service provider to the service customer. A logic expression follows first-order logic. Expressions contain the usual operators -- *and*, *or*, *not*, etc. -- which connect predicates or, again, expressions. *Predicates* can have SLA parameters and scalar values as parameters. By extending an abstract predicate type, new domain-specific predicates can be introduced as needed. Similarly, expressions could be extended e.g., to contain variables and quantifiers. This provides the parties the expressiveness to define complex states of the service.
- A service level objective may have an *evaluation event*, which defines when the expression of the service level objective should be evaluated. The most common evaluation event is *NewValue*, used 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 within a guarantee or a commonly used schedule can be referred to.

Listing 4 illustrates service level objectives.

**Listing 4. Service level objective example**

```xml
<ServiceLevelObjective name="slo1">
  <Obliged>ACMEProvider</Obliged>
  <Validity>
    <Start>2002-11-30T14:00:00.000-05:00</Start>
    <End>2002-12-31T14:00:00.000-05:00</End>
  </Validity>
  <Expression>
    <Implies>
      <Expression>
        <Predicate xsi:type="Less">
          <SLAParameter>Transactions</SLAParameter>
          <Value>10000</Value>
        </Predicate>
      </Expression>
      <Expression>
        <Predicate xsi:type="Less">
          <SLAParameter>AverageResponseTime</SLAParameter>
          <Value>0.5</Value>
        </Predicate>
      </Expression>
    </Implies>
  </Expression>
</ServiceLevelObjective>
```
**Listing 4** shows a service level objective given by **ACMEProvider** for one month in 2002. It guarantees that the SLA parameter **AverageResponseTime** must be less than 0.5 if the SLA parameter **Transactions** is less than 10,000. This condition should be evaluated each time a new value for the SLA parameter is available.

**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 by this kind of guarantee. This particularly includes the supporting parties of the contract. An action guarantee comprises the following elements and attributes:

- The **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 **evaluation event** or an evaluation **Schedule** defines when the precondition is evaluated.
- The **qualified action** 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 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.
- The **execution modality** 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 **always**, **on entering a condition** or **on entering and leaving a condition**.

**Listing 5** illustrates an action guarantee.

**Listing 5. Action guarantee example**

```xml
<ActionGuarantee name="ag2">
  <Obliged>XYZAuditing</Obliged>
  <Expression>
    <Predicate xsi:type="Violation">
      <ServiceLevelGuarantee>slo1</ServiceLevelGuarantee>
    </Predicate>
  </Expression>
  <EvaluationEvent>NewValue</EvaluationEvent>
  <QualifiedAction>
    <Party>ACustomer</Party>
    <Action actionName="notification" xsi:type="Notification">
      <NotificationType>Violation</NotificationType>
      <CausingGuarantee>ag2</CausingGuarantee>
    </Action>
  </QualifiedAction>
</ActionGuarantee>
```
In the example, XYZAuditing is obliged to invoke the notification action of the service customer ACustomer if a violation of the above-mentioned guarantee slo1 occurs. The precondition should be evaluated every time the evaluation of slo1 returns a new value. The notification should always be executed. The action has three parameters: the type of notification, the guarantee that caused it to be sent, and SLA parameters relevant for understanding the reason of the notification.

Workload management

The SLA-based workload management system consists of a collection of runtime control mechanisms that allocate platform resources to Web services requests. The workload management system allocates resources to achieve high-level objectives. We group these objectives into two categories. The first category of objectives describes the needs of customers or users invoking applications or services running on the Web service platform. We use SLAs to describe these objectives. The second category of objectives deals with defining the operational goals or needs for the service provider. For example, pursuing high resource utilization or favoring one type of customers over others falls into this category. We use the term operational goals to describe these objectives.

Request classification

The SLA-based workload management system must explicitly distinguish requests based on their performance objectives. The distinction happens at every level of the system as well as both at the edge and at the core of the Web services platform. We typically group requests in service classes where each service class contains requests with similar performance goals (i.e., SLA with similar performance objectives). Therefore, one of the fundamental requirements for Web services platforms that need to support SLAs is that the core of the platform makes a distinction between service classes. The introduction of service classes into Web services platforms, although not in the Web service standard at this time, can be accomplished in a fully compatible manner. For example, as shown in Figure 5, a classification mechanism running as an Axis handler maybe inserted in the request path. The classification handler will use the authentication parameters carried by the message context together with user subscription data available on the platform to map each request to a specific service class. The classification handler in turn injects the service class information in the classification handler to be consumed by the workload management control mechanisms.

Control mechanisms

Typically, the SLA workload management system uses five kinds of control mechanisms: admission control, policing, flow control, scheduling and routing. These levels of control perform different workload management functions; a given platform may use only a subset of these functions. Figure 5 shows a prototypical deployment of SLA-based workload management mechanisms.
The admission controller regulates the acceptance or blocking of incoming requests on a session-by-session basis. During very high load conditions, this controller implements an admission policy to limit the saturation of the platform and its resources. The admission policy defines the percentage of Web services requests to discard when the load reaches a given threshold. The percentage of requests and the threshold depends on the service class (see [HYM93] for an example of admission control policies).

The policing mechanism controls the request stream during the entire active phase of the session and restricts the behavior of the client to the characteristics negotiated in the SLA. When the request intensities from all sources are below a given threshold, the policing mechanisms can allow an individual client to exceed its negotiated limits. However, when the request load increases above a threshold, the policing mechanisms will trim the load from the client sources that exceed their negotiated limits to prevent platform congestion and performance degradation. Policing algorithms proposed so far include the "leaky bucket," the "jumping window," the "moving window," and the "exponentially weighted moving average" [RAT91]. All policing algorithms take as input a set of parameters that define the maximum size and length of a request burst.

To achieve good performance, we need to control the level of load concurrency on each server. As shown in Figure 6, as the concurrent number of requests handled by a server increases the throughput increases up to a saturation point.
Figure 6. Throughput performance as a function of the number of concurrent requests executing on a server

![Graph showing throughput performance](image)

After this point, the performance degrades because of resource contention, context switching, and thrashing. The flow control mechanism makes sure that each server handles at most W requests concurrently to maximize each server’s throughput. When a the platform load exceeds the maximum number of concurrent requests that each server can handle for optimal performance, the workload management system will buffer all the requests that exceed the limit. As shown in Figure 6, the workload management system queues requests in separate buffers according to requests' service class. The scheduling mechanism determines which requests should be dispatched for execution to the server. Various scheduling disciplines can be used (static priority, FIFO, etc.). A common scheduling discipline is Weighted Fair Queuing (WFQ). When the scheduler implements the WFQ discipline, requests are removed from the buffers for execution in a round-robin fashion. The scheduler associates a weight with each service class and the proportion of requests selected from each buffer is a function of this weight. The router dispatches requests to one of the servers that can execute it. The routing decision depends on the service class of the request and the state of the system (which consists of the utilization of servers and the backlog of work on the servers). Typically, the router uses a set of weights to determine the proportions of requests that should be dispatched to each server.

**Global Resource Manager and the LEAD design pattern for workload management**

Each control mechanism acts on every request, and we have implemented them as Axis handlers. In order to minimize the overhead we build these mechanisms following the LEAD design pattern [PAC95].
Figure 7. The LEAD design pattern for efficiency

![Diagram of LEAD pattern]

Figure 7 shows the LEAD pattern and describes the functional components of a workload management system, together with the interactions among components and with the outside world. The main idea behind the LEAD pattern is that the task of computing a control policy is separated from the task of implementing the control policy on a particular service request. Following this separation, the model contains three types of mechanisms: the SLA legislator, the SLA executor, and the SLA aggregator and distributor. A set of these mechanisms interacts to perform a specific resource control task (e.g., scheduling, routing, policing, etc.). The legislator generates a set of rules, which must be observed when handling a service request. We call this set of rules the control policy. The executor enforces the control policy by applying these rules to incoming requests. In other words, the executor implements the control policy computed by the legislator. The aggregator and distributor reads a set of control policies destined for the same executor (or collection of executors) and aggregates them into one single policy that can be read by the executor.

The executor is driven by external stimuli. Its task is to handle incoming requests. The legislator, in contrast, is either invoked by the executor or runs on its own and periodically re-computes the control policy. The legislator performs its operation usually on a time scale much slower than that of the executor, since the computational complexity of the workload management system resides in the legislator part. Legislator and executor interact by sharing a data object, the control policy, which is written by the legislator and read by the executor.

The interaction between legislator and executor can be either synchronous or asynchronous. In the synchronous case, the legislator invokes the executor, e.g., in the form of a function call. In the case of asynchronous interaction, legislator and executor form a loosely coupled subsystem. Each mechanism runs on its own time scale, and they communicate asynchronously via the shared policy object. The aggregator and distributor collects policy objects from legislators and in turn distributes them to executors. Note that asynchronous interaction between legislator and
executor allows them to run independently and on different time scales. Therefore, they can be optimized according to different requirements: the executor guarantees fast decision times, while the legislator optimizes the utilization of the resource, e.g., by solving an optimization problem.

The LEAD pattern covers a wide range of possible implementation decisions. It covers single-threaded, distributed, and parallel implementations of SLA management subsystems, depending on whether the mechanisms are intended to run on the same or different containers and whether their interaction is designed to be synchronous or asynchronous. The model supports a case in which several executors share the same legislator. In some instances, several legislators may control one executor. For example, in the case of routing, we may have one legislator that optimizes the platform resources to guarantee reliability objectives while another legislator may produce control policies to balance the load of requests among a set of servers. Both legislators produce a control policy consisting of a set of weights. To provide a flexible design that includes such a case we introduce the concept of control policy distributors and aggregators. The aggregator reads a set of control policies destined for the same executor (or collection of executors) and aggregates them into one single policy that can be read by the executor.

In Figure 5, the global manager implements the legislator functions defined by the LEAD pattern. The global manager collects measurement data from the control mechanisms and from resource monitors residing on the servers. The global manager uses SLA parameters and resource configuration information. The system operator and other management mechanisms can change the resource configuration dynamically (e.g., in response to faults, scheduled maintenance, backups, etc.). The global resource manager considers also the operational goals provided by a system operator. The operation goals define the value associated with meeting, exceeding, or failing to meet the target SLO for each class of service. The workload manager analyzes the monitored data and solves an optimization problem for the allocation of resources so as to maximize operational goals.

Summary

A single Web service can be deployed to simultaneously provide different service levels to different clients. Supporting a specific service level and providing assurances to a service client are the keys to successful deployment of Web services for business applications, and for enabling business models such as eHosting and eBusiness-on-Demand. Specification of a service level agreement and how this specification can be used for automated resource provisioning and runtime goal management are explored in this paper. One key issue in SLA specification is making this specification unambiguous so that it can be used to automatically configure the monitor for monitoring SLA performance. The paper also provided an overview of the WSLA specification language that has been supported in the IBM Web Services Tool Kit.

Acknowledgements

This paper has drawn upon the joint work with many of our colleagues. In particular, we would like to acknowledge the contributions of Richard Franck, Alexander Keller, Robert Kearney, and Richard King on SLA specification, creation, and compliance monitoring; of Michael Spreitzer, Asser Tantawi, and Alaa Youssef on goal-based workload management; and of Joe Bockhold,
Seraphin Calo, Paul Chen, Doug Davis, Steve Fontes, Joachim Hagmeier, Dietmar Kuebler, Heather Kreger, Mike Polan, Charlie Redlin, and Steve Roberts on the overall management framework.

© Copyright IBM Corporation 2003
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)