Web services on demand: WSLA-driven automated management

by A. Dan
D. Davis
R. Kearney
A. Keller
R. King
D. Kuebler
H. Ludwig
M. Polan
M. Spreitzer
A. Youssef

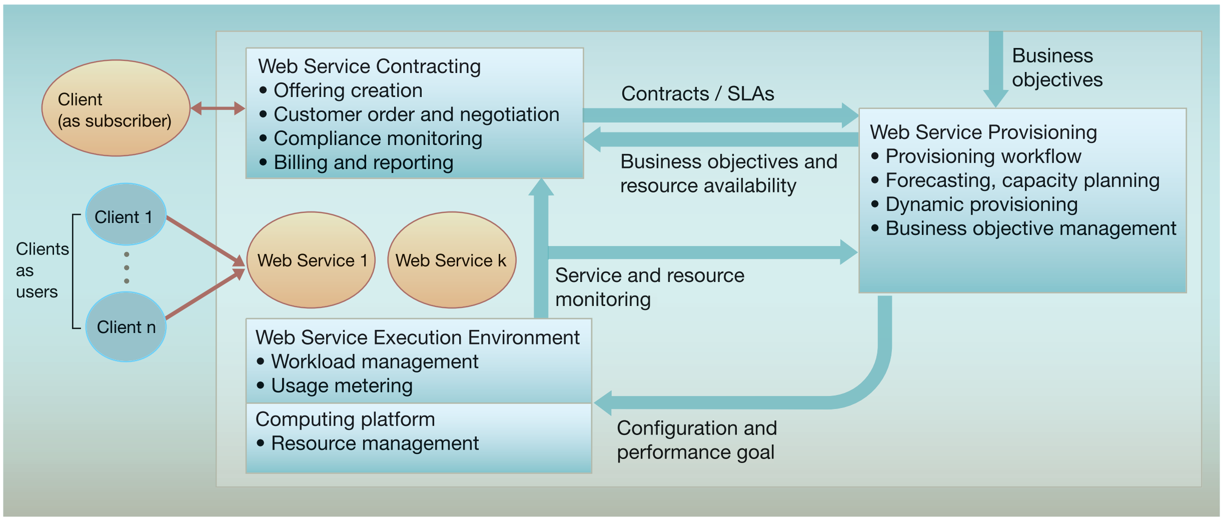
In this paper we describe a framework for providing customers of Web services differentiated levels of service through the use of automated management and service level agreements (SLAs). The framework comprises the Web Service Level Agreement (WSLA) language, designed to specify SLAs in a flexible and individualized way, a system to provision resources based on service level objectives, a workload management system that prioritizes requests according to the associated SLAs, and a system to monitor compliance with the SLA. This framework was implemented as the utility computing services part of the IBM Emerging Technologies Tool Kit, which is publicly available on the IBM alphaWorks™ Web site.

In recent years, we have seen a sharp rise in outsourcing of business applications and processes. The ability to outsource certain business operations allows each business to focus on its core activities and competencies. On the service providers' end, they become more cost-effective by exploiting economies of scale similar to traditional public utilities.[1,2] Hence, the service providers can be referred to as "utility computing providers." In order to meet the requirements of fast changing market conditions (i.e., in order to be effective, efficient, and flexible), we expect outsourcing to evolve into "dynamic outsourcing" of applications and business processes, whereby service consumers determine programmatically the service provider and/or service attributes, that is, quality of service, to use.[3]

Success of dynamic outsourcing and quickly forming new business relationships are, however, dependent on three critical factors. First, to meet interoperability requirements, access to services needs to be based on open and emerging standards for enabling the service-oriented architecture (SOA) model, and in particular Web and grid services. These services may span a wide range of the outsourcing spectrum, including access to business applications, such as financial services, human resources (HR), and enterprise resource planning (ERP), and infrastructural resources, such as storage, computing resources, and application-hosting platforms. Second, the decision to outsource a part of the business process or application is critically dependent on whether a business partner can be trusted to provide an on-time reliable service. To ensure this quality of service, the service client jointly with the service provider should define a service level agreement (SLA) as a part of a service contract that can be monitored by one or both parties. The same service may be offered at different service levels (in terms of responsiveness, availability, throughput) and priced accordingly. Third, to provide fine-grained outsourcing in a cost-effective and on-time manner, it is essential to support *automated management* of the entire life-cycle of the business relationship: creation of service offering, creation of SLAs with possible negotiation, provisioning of applications and environments, and monitoring of SLAs both for dynamic allocation of resources and for compliance. To facilitate this automated management, the SLAs and other agree-

0018-8670/04/$5.00 © 2004 IBM

Figure 1 High level architecture of the Web-services-on-demand utility computing environment



ments need to be specified in machine-executable forms. In addition to SLAs with customers, the business defines additional objectives for managing the utility computing infrastructure. This may include resource arbitration policies across SLAs (in terms of optimization of profits, customer satisfaction, etc.) and strategies for provisioning additional resources. Resource provisioning and workload management in meeting SLAs with clients and other business objectives may differ significantly across service providers, thus reflecting their unique strategies and automated management processes. A less agile utility computing provider may depend on fixed allocation of resources, whereas a more efficient provider can dynamically redistribute resources across multiple customer workloads.

This paper presents a framework for providing differentiated levels of Web services to different customers through the use of automated management and service level agreements (SLAs). The execution environment is referred to as a "Web-services-on-demand" environment because it exhibits two important on demand characteristics. First, it supports dynamic outsourcing (on demand) where service customers can dynamically form outsourcing relationships with providers, and the execution environment supports the required functionalities for managing the SLA life cycle. Second, to be efficient and profitable, the service provider provisions resources dynamically (i.e., on demand). *In this paper we use the terms "Web services" and "utility computing services" interchangeably.*

Figure 1 illustrates a high level architecture of our implementation, in which functions are grouped into three components: Web service contracting, Web service provisioning, and the Web service execution environment. The Web service execution environment includes the computing platform and workload management.

All customer interactions other than the service invocation itself are managed by Web-service contracting. The utility computing provider creates various service offerings that are defined in terms of service levels (fixed or negotiable), rating (pricing), and various restrictions that may be placed on the usage of this service. Tools for creating an offering may take into account various business objectives, resource availability, as well as considerations on the profitability and the difficulty of supporting such an offering. A customer order (also referred to as a subscription) for an offering results in the creation of an SLA as part of a contractual agreement. The SLA is expressed via the Web Service Level Agreement (WSLA) language.[4] For flexibility, certain terms of the contract can be negotiated.[5,6,7] The subscription is tracked during the fulfillment process to make sure that the service level guarantees agreed upon in the SLA are adhered to, a process referred to as compliance monitoring. Data on service and resource usage, as well as data on any violation of service level guarantees, are used in billing and reporting.

Processes for Web-service provisioning and resource allocation are typically hidden from the customer. As shown in Figure 1, Web-service provisioning takes as input both the customer SLA and the business objectives. The multiple-step process of Web-service provisioning is performed as a workflow. In order to optimize on service cost, the utility computing system may not allocate all required resources in advance. Instead, it may use sophisticated online-forecasting and capacity-planning tools to plan this provisioning. In a complex environment, characterized by many continuously varying customer workloads and/or a complex resource configuration with hard-to-estimate capabilities, dynamic resource provisioning may be necessary to fine-tune resource allocation. Resource reallocation takes into account various business objectives, such as profit optimization and customer satisfaction (e.g., no more than a certain number of violations during a given period, independent of penalties). As illustrated in Figure 1, the resource allocation plan affects the setting of configuration and performance goals, which, in turn, affects the way Web-service invocations are managed.

A Web-service invocation goes through some preliminary steps before service is rendered by the Web-service execution environment (Figure 1). These include authenticating the requester, identifying the contractual agreement and its SLA, and setting up performance goal management (managing response time and throughput by prioritizing execution of service requests from multiple customers). The service usage data and the actual performance data are collected online for checking compliance with an SLA and for customer billing and reporting.

The above framework was implemented by integrating various technologies that were developed by teams in the IBM Research Division and the IBM Software Group. A subset of the framework is publicly available on the IBM alphaWorks* Web site as the utility computing services component of the IBM Emerging Technologies Tool Kit (ETTK).[8] The ETTK utility computing component includes most of the features of SLA life-cycle management described in this paper (except dynamic provisioning of resources): Web service contracting, WSLA specification and monitoring, and the SLA-based workload management.

The remainder of the paper is organized as follows. In the next section we discuss SLAs in Web-service contracts and provide an overview of the WSLA-based specification. The following three sections describe in more detail the three major components of the architecture: Web-service contracting, Web-service provisioning, and the Web-service execution environment with workload management. We provide a summary and final comments in the last section.

SLAs in Web-service contracts

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,[9] establishing a generic framework for service level management in cross-organizational environments remains a challenge. In this section, we illustrate via a utility computing scenario the detailed SLA specification requirements for enabling the monitoring of SLA compliance, and how these requirements are addressed in the WSLA specification and the WSLA monitoring framework. We also describe our approach to providing fine-grained and flexible accounting for utility computing.

Example scenario: utility computing services for financial institutions. A financial institution, FINANCE, offers a suite of Web services that provides functions for portfolio management. These services are used remotely by customer applications. The customers are billed only for services used. A typical FINANCE customer is SMALLBUS, a small business that provides portfolio management portals for use by its employees. The portlets[10] that make up the portal consume the Web services from FINANCE.

FINANCE, rather than acquiring and owning the infrastructure for hosting these services, seeks out a provider of such services. Utility service provider UTILSERV supplies to FINANCE the computing resources needed to host the portfolio management Web services, which include servers, storage systems, networking components, and Internet connectivity. UTILSERV also provides management services such as backup, restore, security, and provisioning. An SLA between UTILSERV and FINANCE documents the characteristics of the provided infrastructure, such as network throughput, or the maximum CPU utilization of the processors used to run the portfolio management Web services.

The contractual agreement between FINANCE and SMALLBUS includes an SLA that specifies performance objectives such as response times and throughputs. These objectives are based on the SLA between FINANCE and UTILSERV. For example, a stock purchase transaction involves a confirmation delay that is part of the SLA and depends on the SLA between FINANCE and UTILSERV.

FINANCE provides a portal service to SMALLBUS that includes portlets from FINANCE and uses the infrastructure provided by UTILSERV. An SLA between SMALLBUS and FINANCE describes the expected behavior of the portal.

Rating models are used to bill consumers for services. In the case of UTILSERV and FINANCE, the rating model includes the following properties and their values.

Base_price = 10$ per month
Storage_price = 2 $ per GB average per day
Network_bandwith price = 3 $ per Mb per hour
Internet_connectivity_price = 10 $ per GB volume per month
Internet_connectivity_exceed_price = 2 $ per 10 MB volume
Network_data_throughput_violation_price = 1 $ per 0.01% deviation

For FINANCE and its customers, the rating model includes:

Stock_purchase_transaction_service_price = 5 $ per 1000 $ transaction volume
Stock_purchase_limit_price = 3 $ per transaction
Response_time_violation_price = 2 $ per 0.01 % deviation
Throughput_violation_price = 1 $ per 0.01 % deviation

The overall contract between the UTILSERV and SMALLBUS includes service level guarantees, the penalty upon violation of each guarantee, and the rating on use of this service.

Contracting for Web services. The way computing utilities are contracted is fundamentally different from application hosting as practiced in recent years. An application hosting service involves dedicating resources to that service. This implies a fixed cost to the provider for providing and managing these dedicated resources. The cost is typically passed on to the service customer as a recurring charge, regardless of the load on these resources and, thus, the extent to which the customer makes use of these resources. The arrangement is inflexible for the service provider and, as a result, unduly expensive for the customer.

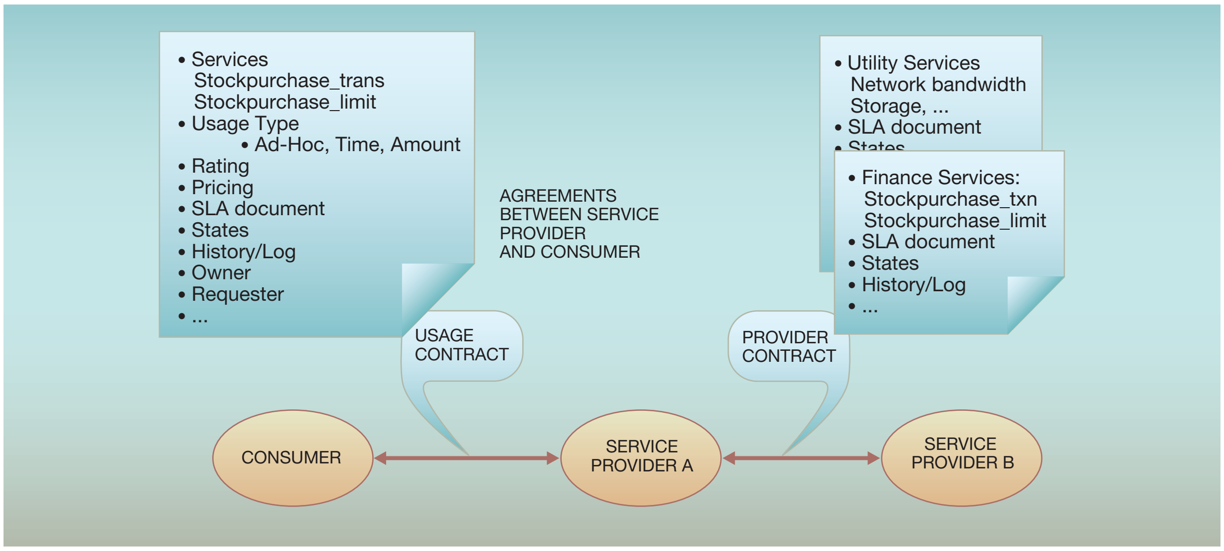
Consider a public utility model for service delivery, such as electric power. Electricity is always available when needed and customers are billed only for the power they actually consume. SLAs in utility contracts dictate the expected reliability of the power delivery and the maximum power a customer may draw. The customer load varies in time, so the utility service provider is able to allocate power dynamically to minimize costs and meet the SLAs. A utility computing provider can operate using the same principle. Computing resources can be added or removed from the service in order to meet current or anticipated load. Although this is perhaps easiest to visualize in terms of computing power, where processors could be added or removed as needed, utility computing services are not limited to this type of resource.

Utility computing services are therefore provided on demand, following a public service utility model. The contract between customer and provider defines charges for services at a relatively fine grain, e.g., processor minutes used rather than allocated computing servers per year. The terms and conditions of the delivery of the service are defined in a contract, including the definition of the service, the rating model (specifying how the service is priced), and the performance guarantees on various aspects of the service.

Figure 2 illustrates the contractual relationships in a utility computing environment. Service provider B provides services and establishes an “internal” contract with service provider A, which is referred to as a *provider contract*. For service provider A, service provider B is sometimes referred to as a “supplier.” The provider contract between UTILSERV and FINANCE covers services for storage, network bandwidth, and the like, and includes an SLA document describing the properties of the infrastructure provided.

Service provider A offers its services under certain terms and conditions to the consumer. When the consumer subscribes to this service an “external” contract is created, which is referred to as the *usage contract*. As illustrated in Figure 2, the usage contract contains definitions of the offered services. The usage type specification includes an ad hoc type that counts the number of requests, whereas the “time” field is used to accumulate service request time, and the “amount” field is used to accumulate measure-

Figure 2 Contractual relationships in a utility computing environment



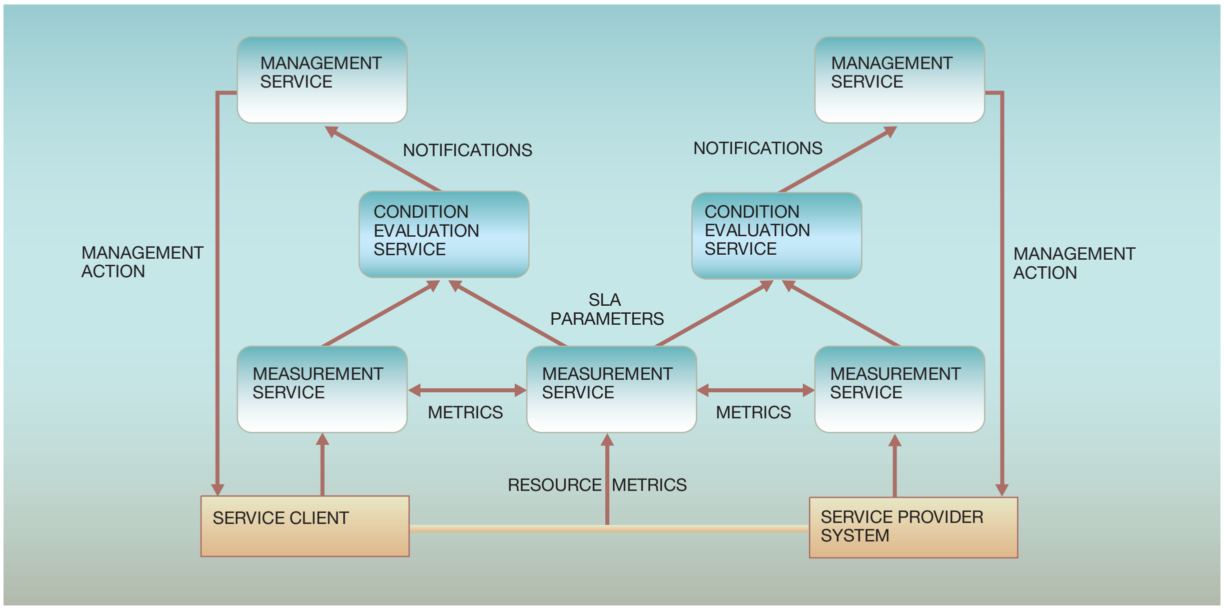
ments, such as “KB per second.” The “requester” field names the customer, SMALLBUS in our scenario.

Service provider A provides services for the consumer in Figure 2. At the same time, service provider A is a consumer for the services provided by service provider B. Service provider A may resell to the consumer the services provided by service provider B if the provider contract allows it, e.g., through a license agreement.

SLA compliance monitoring. An SLA defines the agreed level of performance for a particular service between a service provider and a service customer.[11] Having analyzed a number of SLAs used throughout the industry for typical application service provisioning, Web hosting, and IT outsourcing scenarios, we found that many SLAs contain a common set of key elements: the names of the two parties, the SLA parameters, the metrics used as input and the algorithm for determining the values of the SLA parameters, and the service guarantees and the appropriate actions to be taken if a violation of these guarantees is detected. The requirements on the scope and expressiveness of SLAs are as follows:

- In the Web services context, service guarantees and the relevant input parameters must be associated with individual Web services, as defined in a Web Services Description Language[12] (WSDL) file, or processes of Web services as defined in a BPEL4WS[13] specification. Service level guarantees need to be defined for individual operations in bindings. Definitions on the port type level are only of limited use. For example, response times can only be expressed in a meaningful way on a per-operation level because the processing of different operations in the application service takes a different amount of time, independent of the current load. Different bindings may also yield different response times for the same operation.
- The number of different SLA parameters that can be defined for a service is potentially very large. Even seemingly simple parameters, such as response time or throughput, can be defined in many different ways: from the client, the application server, or the application point of view; sampled or averaged; the time interval over which the average is computed; and averaging for the operation type or for each operation individually. An SLA must provide a description how SLA parameters are measured and aggregated from resource metrics (metrics that are retrieved directly from managed resources).
- The SLA must be able to express a large variety of contractual obligations. This includes service level objectives with respect to SLA parameters as well as a deterministic set of actions if violations occur or other critical situations arise.

Figure 3 Services involved in SLA-compliance monitoring with multiple parties



- SLA monitoring may require the involvement of third parties. They come into play when either a task needs to be performed that the two parties are not willing to commit to, or when a third party is preferred for reasons of trust. Third parties offer objectivity in the SLA monitoring process and help avoid disputes.[14]
- Because the task of computing SLA parameters and evaluating contractual obligations may be further split among multiple agents, it is important that each one of these agents receives only the part of the contract that it needs to know to carry out its tasks. Consequently, a mechanism to send only the relevant parts of the SLA to third parties must be provided.
- The SLA must be formally defined for automatic processing. This is important for the automatic provisioning of a service and for the automatic setup of the SLA monitoring and management infrastructure.

Web Service Level Agreements. During the contracting process, after the main elements of the SLA are agreed upon, customer and provider may define third parties (in the WSLA context we refer to these as *supporting parties*) to which SLA monitoring tasks may be delegated. Keynote Systems, Inc.[15] and Xaffire, Inc.[16] (previously Matrix NetSystems) are examples of such third-party-monitoring service providers.

When the SLA is finalized, both provider and customer make the SLA document available for deployment. The *deployment service* is responsible for checking the validity of the SLA and distributing it either in full or in part to the supporting parties.

Figure 3 illustrates the services involved in compliance monitoring when multiple parties are involved. Services that may be outsourced to third parties are either measurement services or condition evaluation services.

- The *measurement* service maintains information on the current system configuration and runtime 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 from outside the service provider's domain, for example, by probing or intercepting client transactions. A measurement service may measure all or a subset of the SLA parameters. Multiple measurement services may simultaneously measure the same metrics, e.g., a measurement service may be located within the provider's domain while another measurement service probes the service offered by the provider across the Internet from various locations. As depicted in Figure 3, measurement services may be cascaded, that is, a third measurement service may be used to aggregate data computed by other measurement services. For our discussion, we refer to metrics that are retrieved directly from managed resources as *resource metrics*. *Composite metrics*, in contrast, are created by aggregating 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 (e.g., top 5 percent, minimum, maximum, mean, median etc.). This is usually done by a measurement service within a service provider's domain, but can be outsourced to a third-party measurement service as well.

- The *condition evaluation service* is responsible for monitoring compliance of the SLA parameters at runtime with the agreed-upon service level objective (SLO) by comparing measured parameters against the thresholds defined in the SLA and notifying the management services of the customer and the provider. It obtains measured values of SLA parameters from one or more measurement services and tests them against the guarantees given in the SLA. This can be done each time a new value is available, or periodically.

Finally, both service customer and provider have a *management service*. Upon receipt of a notification, the management service (usually implemented as part of a traditional management platform) takes appropriate actions to correct a problem, as specified in the SLA. 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.

The WSLA Language Specification. The WSLA Language Specification[4] defines a type system for the various SLA artifacts. It is based on the XML Schema. In principle, there are many possible types of information and rules that can be included in an SLA; however, there is consensus on the general structure of an SLA. WSLA embraces this structure by dividing an SLA into three sections: parties, service description, and obligations.

The *parties* section identifies the contractual parties and contains the relevant technical properties, such as their network addresses and interface definitions (e.g., the ports for receiving notifications). A party can be a *signatory* party or a *supporting* party. The information for a supporting party includes, in addition to the information for a signatory party, an attribute indicating one or more sponsors of the supporting party.

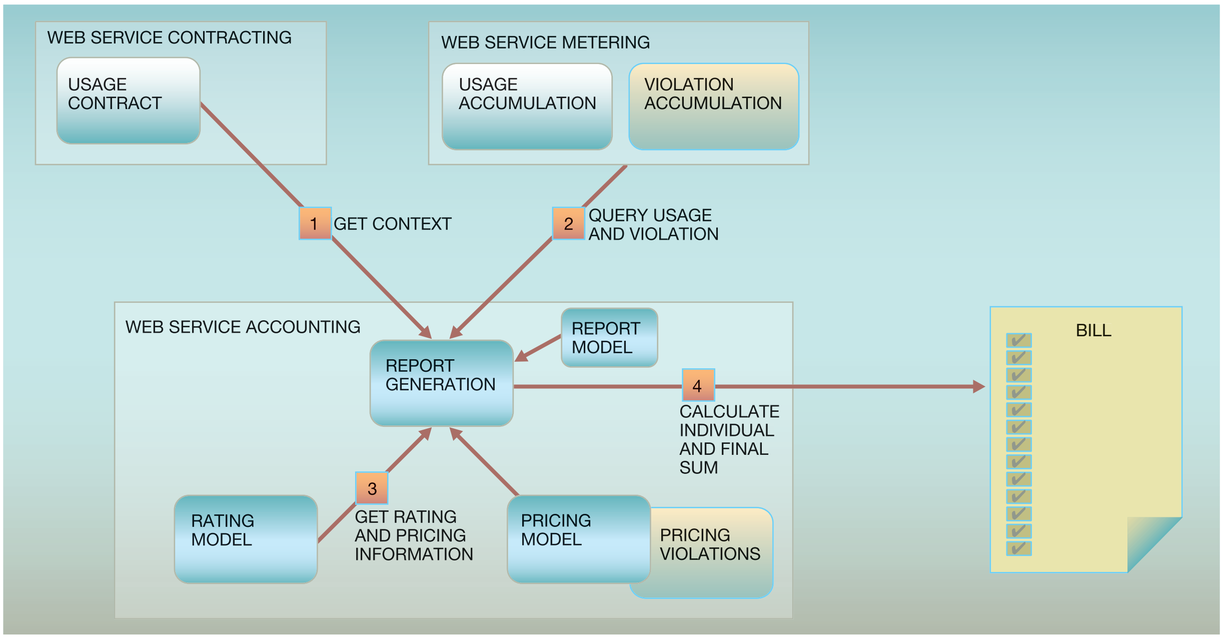
The *service description* section of the SLA specifies the characteristics of the service and its observable parameters. For every service operation, one or more *bindings* may be specified; a binding is the transport encoding for the messages to be exchanged. In addition, one or more SLA parameters of the service may be specified. Examples of such SLA parameters are service availability, throughput, or response time.

Every SLA parameter refers to one *metric*, which, in turn, may aggregate one or more other (composite or resource) metrics, according to a measurement directive or function. Examples of composite metrics are maximum response time, average availability of a service, and minimum throughput. Examples of resource metrics are: system uptime, service outage period, and number of service invocations. *Measurement directives* specify how an individual resource metric can be obtained from a managed resource or from a system acting as a proxy for the resource. Typical examples of measurement directives are: the uniform resource identifier of a hosted computer program, a protocol message such as an SNMP (simple network management protocol) GET message, a command for invoking scripts or compiled programs, and a query statement issued against a database or data warehouse. *Functions* are the algorithms that specify how a composite metric is computed. Functions can be formulas of arbitrary length containing mean, median, sum, minimum, maximum, and various other arithmetic or statistical operators, or time series constructors. For every function, a *schedule* is specified. It defines the time intervals during which the functions are executed to retrieve and compute the metrics. These time intervals are specified by means of start time, duration, and frequency. Examples of frequency specifications are weekly, daily, hourly, and every minute.

Obligations, the last section of an SLA, define the SLO, that is, guarantees and constraints that may be imposed on the SLA parameters. (For an in-depth discussion of the WSLA language and usage examples, the reader is referred to References 4 and 17.)

Accounting. The utility model for service delivery discussed at the beginning of this section, introduces

Figure 4 Functional overview of web service accounting



the need for flexible accounting schemes that support billing for actual usage. The accumulated usage and the number and severity of SLA violations (detected by the SLA compliance monitoring previously described) are the input parameters for generating service usage reports and bills. The specifics of service rating and billing are defined in the usage contract (see Figure 2). This document contains all details of the overall agreement between customer and provider, and it is used by other components, such as usage metering.

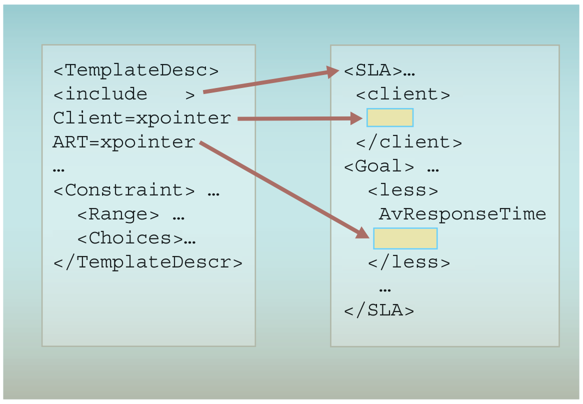
Figure 4 shows the functional overview of the Web-service accounting component (shown in Figure 1 as the billing and reporting function of Web Service Contracting). Web-service accounting gets the context from the appropriate usage contract (step 1) and queries Web-service metering for usage accumulation and service violations (step 2). It then obtains rating and pricing information for the services in the usage contract (step 3), calculates the respective single events, and produces a report, and possibly a bill, in a form specified by a report model. Web-service accounting uses two database tables: one defines a key associated with a rating model and the currency that should be used; the other defines the properties of the rating model, such as stock purchase transaction price, stock purchase limit price, response time violation price, and throughput violation price. The pricing for each property is then defined in a separate column. Thus, service providers can introduce their own accounting schemes by defining rating models through the use of properties, implement these rating models, and associate the usage contract with a pricing model.

Creating offerings and managing subscriptions

In this section, we present the Web-service contracting component, which manages all relevant aspects of contracts for Web services. The approach we take is extensible in that it has the capability to associate and treat multiple related contracts. As a consequence, even if a contract is already in place, it can accept new features.

For runtime processing the contract documents are represented as *contract objects*. The system comes with a set of predefined contract object types as outlined in the next subsection. We describe the processes operating on these contract objects for creating offerings and managing subscriptions. The framework we describe also includes persistent stor-

Figure 5 Illustration of a template



```
<TemplateDesc>
<include   >
Client=xpointer
ART=xpointer
…
<Constraint> …
  <Range> …
  <Choices>…
</TemplateDescr>
```

```
<SLA>…
 <client>

 </client>
<Goal> …
  <less>
   AvResponseTime

  </less>
   …
</SLA>
```

age and retrieval of contracts as well as contract validation methods during runtime processing.

Contract object types. The Web-service contracting framework supports arbitrary contract object types; each contract object is an instance of one contract object type. All contract object types are based on the same data model. All contract objects contain:

- An *owner* who is responsible for the administrative management of the contract
- A *state* that can be active, inactive, or deleted. Following the creation of the contract object, the state is set to inactive. This allows exploiters to add functionality, such as creating contract objects in advance and making sure they represent a new offering. A contract object is never deleted from the database (regardless of the term "deleted state") because of legal and audit considerations.
- A reference denoting the contract object type
- A contract document, an XML structure that contains the customer contract data. The format of the contract document varies; any kind of contract description can be used: string, XML document, even binaries as Java** strings.

The Web-service contracting framework implemented in ETTK provides four basic contract object types.

1. *Basic*: This type is used for storing general purpose contract objects that have no relation to other contract objects, for example, SLA basic templates (see the section "SLA templates"). All other contract object types have relations to other nonbasic contract object types. Contract objects of this type are only represented by the contract document.
2. *Provider contract*: Enables the aggregation of services from service suppliers (for a service provider, its own providers are known as suppliers). It is possible to create logical units (e.g., a provider contract logical unit called supplierOfStockPurchaseServices) that aggregate all services from the same supplier. Both, service supplier and service provider can add services to provider contracts. This contract object type can be extended to cover internal cost.
3. *Offer*: This type is used by the service provider as a template to create offerings. It can be empty when created and filled in as needed, for example, during contract negotiation. To allow full flexibility, different service operations may be linked to different rating models.
4. *Usage contract*: This contract object type represents the instance of a consumer contract for a particular service. All further service-related actions are prescribed by this contract. The service operations in the usage contract are linked to the provider contracts that aggregate the service operations.

In order to support extensibility, the Web-service contracting framework allows adding properties to existing contract types. Furthermore, if needed, new contract object types can also be added. In the latter case, a plug-in is required that specifies how to access, update, and delete the newly defined contract object.

SLA templates. A WSLA template associated with an offering defines the quality-of-service properties of the service. Conceptually, a template is a WSLA document that contains fields to be filled in during the subscription process. In addition, a template contains a set of constraints on the fields that express the quality of service (QoS) guarantees associated with the service. For example, a field representing the transaction rate might only accept values between 10 and 1000. For internal processing purposes, templates can be associated with guides that implement the function to fill in a value of a field. This specification is not made available to service consumers.

A template for an SLA (or for any XML-based document) consists of the following parts (see Figure 5):

1. One (or more) partially-filled XML documents (forms) that prescribe the general outline of the final document. For our purposes, this set consists of a single WSLA document with the structure of an SLA and with values for certain fields left blank. These values, such as customer name, SLA name, throughput, and response time, will be inserted during the authoring process.
2. An associated XML document describing the fields that can be updated, and the rules for updating these fields.
    a. A list of named XPointers (pointers) identifying both the file, and the location (field) within the file, from which some value is to be set.
    b. For each pointer in the above list, an optional list of constraints placed upon the value. For example, a constraint may identify the maximum and/or minimum values for the field, or an enumeration of the set of possible values for the field.

The following is an example of a pointer:

```
<Pointer form="ID1025194360741.1"
         id="ID1025194425114.3"
         name="Value"
         xpath="#xpointer(/wsla:SLA/wsla:
                                    Obligations/wsla:
         ServiceLevelObjective[1]/wsla:Expression/wsla:
         Implies/wsla:Expression[2]/wsla:Predicate/wsla:
                                              Value)">
   <Restrictions>
      <minInclusive value="10.0"/>
      <maxInclusive value="1000.0"/>
   </Restrictions>
</Pointer>
```

WSLA templates associated with offerings are either created case by case or, as implemented in the ETTK, refined from a base template by adding information from the WSDL definitions of the provider contracts, and additional performance parameters. Here, "form=" and the "id=" are identifiers of the form being referenced, and of the pointer *id*, which may be used for associating internal implementation details (see below). At a later stage in the subscription life cycle, the WSLA template for an offering is finally filled out completely, and thus a WSLA instance is created.

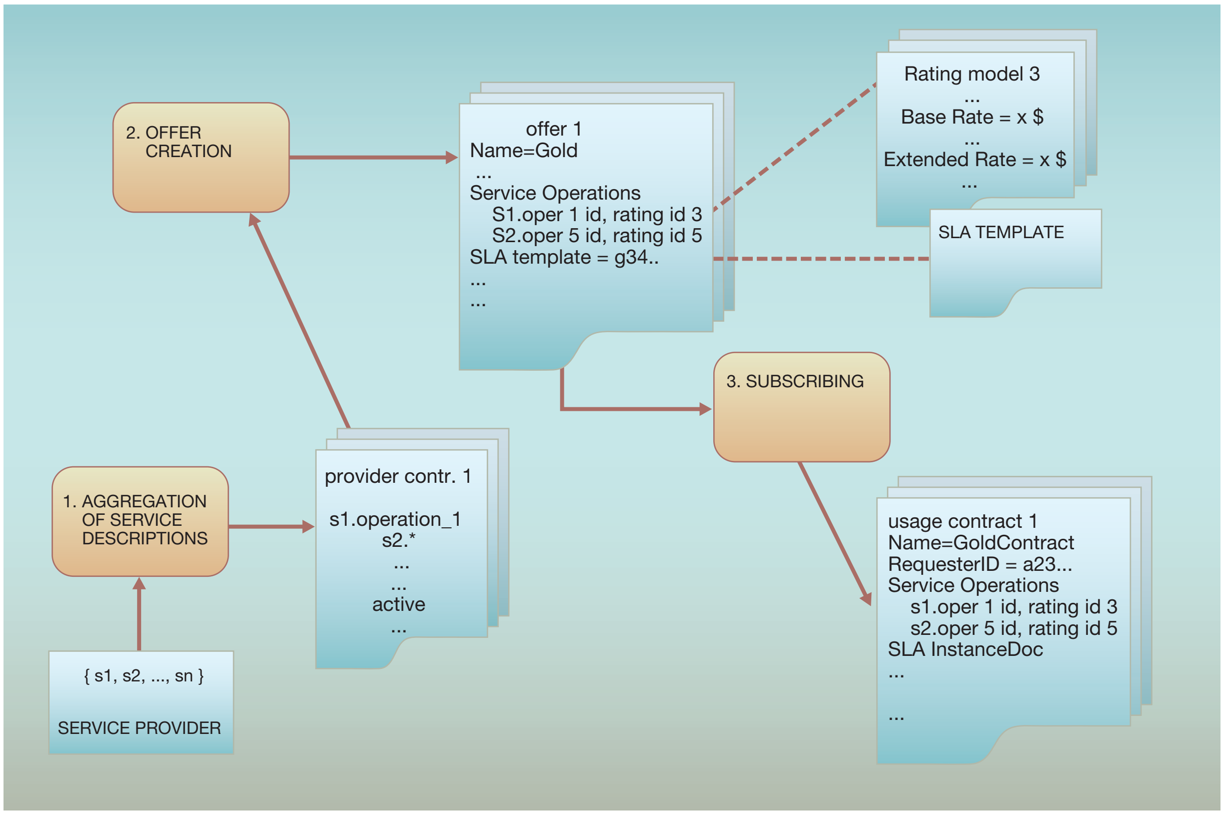
The above template structure is further extended for aiding the authoring process, and this extended template is referred to as the *implementation template*. It includes a set of "guides" (implemented as Java classes) that will be executed during the authoring stage. A guide associated with a pointer returns a String value that will be inserted into the location specified by the pointer. Because each guide is a Java class, it is capable of obtaining data from a broad range of sources. The guides are grouped into named sets called "modes," such that only the guides associated with this mode are executed in sequence during authoring in that mode. This is useful in order to stage the template refinement process, i.e., creating the offering and registering the subscription. Here is how a guide is described in a template:

```
<Guide cloneable="false"
      name="Insert Average Response Time"
      pointer="ID1025194425114.3">
   <Class>com.ibm.wsla.authoring.guides.
                           ConstantValueGuide</Class>
   <Parameters>
      <Parameter>
         <Name>Inserted value,
                              ref or property:</Name>
         <Value Type="Property">
         <[CDATA[@@AverageResponseTime]]></Value>
      </Parameter>
   </Parameters>
</Guide>
```

The authoring tool, which is implemented as a Java class and encapsulated in a Web service, is easily extended for particular applications. In the current context, there are two methods associated with the two authoring modes, that is, creating an offering and accepting a subscription, which supply the values to be inserted into the SLA template. In both cases, the values are inserted in a dictionary with well-known keys. Each method results in the execution of an associated mode, which itself consists of a set of guides that have access to the keys. Thus, a guide interrogates the dictionary, obtains the appropriate value, and returns that value to the authoring tool for insertion into the SLA. Other guides that are executed as part of a mode are responsible for exporting the SLA into a form suitable for Web-service contracting. In addition to simply inserting values into locations within a form, the authoring process can perform more complex functions, such as conditional execution of modes, iteration over modes, and merging of templates (such as merging a template of a component of an SLA into a template of an SLA).

Web-service contracting processes. Next, we take a look at the Web-service contracting processes in

Figure 6 Web services contracting processes and contract objects



the ETTK that use the previously described contract object types. They are depicted in Figure 6.

The provider contracts are obtained as a result of aggregating the service descriptions from the service provider (and possibly suppliers); this is shown as step 1 in Figure 6. The service provider then creates offerings (step 2) from the description of services in provider contracts and additional offering properties: rating, SLA, and so forth. When a customer subscribes to an offering (step 3), a usage contract is created that documents the subscription. These steps are further described next.

Aggregation of service descriptions. In this process the Web-service descriptions are aggregated (enrolled) in provider contracts. This can be done automatically by using the WSDL descriptions of these services. A logical grouping allows the service provider to maintain control over his service suppliers. Both, the service provider and the service supplier can register services in provider contracts. Here is an example of a provider contract for FINANCE:

```
Name=StockPurchaseServices
Owner=TradingDepartment
State=ACTIVE
Type=PROVIDERCONTRACT
CONTRACTDOCUMENT="
<FinancialServicesSupplierProviderContractDoc>
<services>
<service serviceuuid="id1"
            serviceoperation="getRealTimeQuote"/>
<service serviceuuid="id1"
                  serviceoperation="getAskPrice"/>
<service serviceuuid="id2"
                      serviceoperation="setLimit"/>
<service serviceuuid="id3"
                 serviceoperation="buyAtbestPrice"/>
</services>
</FinancialServicesSupplierProviderContractDoc>"
```

Here, *id* 1 through 3 are internal unique representations of services such as StockPurchaseServices and StockPurchaseTransactionService. This allows us to aggregate multiple services with the same name, which is particularly useful when services from multiple suppliers are used.

Creating an offering. When creating an offering, the provider determines which services from its portfolio will be made available to consumers. This is not only a technical process, as it involves taking into consideration the business objectives, and hence rating and pricing of the offering is essential. Here we focus on the technical aspects of creating the offering as implemented in ETTK, including the association of service, SLA, and rating.

An offering may include services, or service operations, from all available provider contracts. The only prerequisite is that the provider contracts involved must be active. This concept can be used to guarantee that a service is available before it is actually offered. Moreover, contract objects of all types can be created in advance. An offering may include service operations from more than one provider contract.

Each service operation can be associated with a rating model. The rating model defines attributes and prices that will be included in the offering. The identifiers that link the rating models to services/operations are stored in the usage contract. The rating models are external to Web-service contracting. An SLA template is created using an authoring process, and either it is stored with the offering, or, if deployed externally, a reference to the document is kept within the offering.

The use of the contract object type "offering" is optional in the Web-service contracting framework. Implementations other than ETTK may have their own offering creation process and may use only provider and usage contracts.

FINANCE might create an offering as shown next. In this example, servicekey represents a service operation such as StockPurchaseService.getRealtimeQuote, or StockPurchaseService.getAskPrice, whereas ratingkey is the link to a rating model.

```
Name=StockPurchaseServiceOffering
Owner=TradingDepartment
State=ACTIVE
Type=OFFER
CONTRACTDOCUMENT=
"<StockPurchaseServiceOfferContractDoc
name="Stock Purchase Offer"
startdate="2003-02-20 00:00:00.000000000"
enddate="2004-02-20 00:00:00.000000000">
<services>
<service servicekey="1ae01625-03d5-41a5-8917-
                37182a173c99" ratingkey="id1"/>
<service servicekey="2b3e03ef-15ce-4aee-8556-
                35e433b70acb" ratingkey="id2"/>
</services>
</StockPurchaseServiceOfferContractDoc>"
specialOffer=GOLD
AppliesForNewCustomer=YES
SLAtemplate="<wsla>.... The WSLA template
  </wsla>"
```

Note, that the Web-service contracting framework allows extending all default contract object types by adding properties; for example, specialOffer and AppliesForNewCustomers are added properties (value GOLD indicates a premium offering).

Subscribing to an offering. Customers agree on the terms and conditions defined in an offering by subscribing to it. The subscription process results in the creation of a usage contract that represents the subscription, and it is used for all further processes as the context to this contractual relationship.

In the course of the subscription process, a customer retrieves the QoS metrics offered by the provider, aggregates and combines them into various SLA parameters, defines service levels for every SLA parameter, and submits the SLA to the service provider for approval. A business entity carries out the negotiation on behalf of each signatory party (a party that signs the SLA). The business entity embodies the business knowledge, goals, and policies of the party it represents. Such knowledge enables the business entity to determine the service levels that should be specified in the SLA in order to ensure compliance with the business goals. A typical example on the customer side is to define thresholds for response times or throughput according to the price the customer is willing to pay. On the provider side, typical business actions are to decide whether the SLA is acceptable as a whole or whether the customer-specified thresholds are too restrictive. Business entities are typically implemented as automated systems in simple cases or are realized as processes involving humans (especially for negotiating contracts).

In the main, the usage contract incorporates the properties of the offering. In addition, the usage contract contains the unique identity of the subscriber. It serves as the handle to the identity context of the customer, for example, by accessing the digital signature stored in the public key by means of the issuer certification authority and the distinguished name. In this scenario the customer selects the SLA attributes, which are reflected in the SLA document. Contracts in the Web-service contracting framework usually have two parts: the business part (i.e., the contract description that can be signed and/or encrypted) and the part containing technical attributes. This is an important feature, especially for legal purposes, because it ensures the contract cannot be changed after the subscription process concludes. This is similar to the mechanism used to prevent changes to PDF files, which is often used in electronic contracts today.

Web service provisioning

The Web services delivered on demand by a successful utility computing provider must be based on automatic provisioning. In the future, SLA-driven automatic and autonomic[18] provisioning will allow the service provider to allocate resources where and when they are needed, minimizing delivery costs. There is no doubt this will be a complex task, as it will necessarily include components, such as network connectivity, operating system, application server, middleware, and application installation and configuration. Because most of these provisioning operations are beyond the scope of this paper, only those directly related to SLA management will be discussed here.

Types of provisioning. Depending on infrastructure capability, types of services, and business models (e.g., assumption of risk, optimization objectives) one or more of the provisioning scenarios described next may be appropriate for a specific service provisioning. The objective of provisioning is to allocate sufficient resources in order to avoid violations of service level guarantees. In our sample scenario, the hosting service provider can use all the models described below to allocate servers, storage, network bandwidth, and connectivity.

Provisioning scenarios can be classified into three categories.

- *Dedicated resource provisioning:* The simplest case of provisioning is that of dedicated resources allocated to a single consumer. When a subscription request is received, resources are provisioned after considering the service specifications versus the anticipated load. In this case, the SLA system monitors for service level compliance, reporting violations when they occur. In the absence of the capability to add resources dynamically, the dedicated environments are typically over-provisioned anticipating the worst-case workload scenario.
- *Per-SLA virtualized resource provisioning:* In an environment in which consumed resources are virtualized (and underlying physical resources are shared in support of multiple SLAs), the provisioning system evaluates whether an additional SLA can be supported with the resources available to this virtualized system. The system may also add new resources to this shared pool based on the aggregate anticipated load of all supported SLAs.[19] As before, the SLA system monitors for compliance and reports violations.
- *Dynamic resource provisioning:* In this, the most sophisticated on demand environment, resources are allocated to services, as needed. An initial set of resources is provisioned for a service, and the SLA system monitors the performance of that service. When the load changes, new resources are allocated or deallocated dynamically, in order to minimize service delivery costs while meeting SLA objectives. Dynamic provisioning can be used in conjunction with the previous two scenarios for initial resource allocation.

In the preceding example, the agreement between the utility provider UTILSERV and the service provider FINANCE could well be a "dynamic resource provisioning" scenario. The utility provider would allocate an initial set of system resources to support the FINANCE service. As load from SMALLBUS customers increased, additional resources would be provided by UTILSERV to ensure compliance with any response time agreement with FINANCE. When load on the FINANCE service decreased (e.g. after close of business), all but the minimum resources needed to support the FINANCE service could be reallocated to other services offered by UTILSERV or UTILSERV's other customers. Note that in all three scenarios, certain provisioning steps may be performed in advance before the creation of a new SLA, whereby a customer is assigned to a pre-provisioned service with additional provisioning steps based on the information in the new SLA.

Next, we describe the "elements" typically provisioned in addition to the actual resources. Then we describe the configuring of an SLA monitor for monitoring new SLAs. We describe the certification and validation process for ensuring that the provisioned resources meet the SLA requirements. We finally provide an overview of the dynamic provisioning process.

Provisioned elements. In addition to the resources themselves, the following components of the infrastructure must be provisioned to support an SLA-managed system:

1. The following information is required for the SLA management system:
    a. Information about the new subscriber—at a minimum, the identity of the service users.
    b. The service to be consumed by the subscriber.
    c. The SLA objectives associated with consumption of the service. In the case of composite services (when a consumed service itself is made up of underlying SLA-managed services), the SLA objectives of the underlying services must also be known.
    d. A service profile that specifies the actions that the SLA management system must take to respond to anticipated loads.
    e. The provisioning operations specific to the new service. These are the actions that the SLA management system uses to adjust the resources allocated to the service.
2. Configuration of the instrumentation of the service and consumed resources; this is used to provide metrics about the service delivery to the SLA system.
3. Configuration of a measurement service that correlates the measurement data with the service subscription (and the SLA) before delivering it to the SLA systems.

In our example, FINANCE is a subscriber of infrastructure services offered by UTILSERV. Identity information needed for FINANCE includes a profile and generic identity as well as specific identities of members of FINANCE that will administer the FINANCE service. Services consumed are infrastructure resources needed to run the FINANCE service. The SLA objectives reflect the level of service that FINANCE wishes to provide to its customers, SMALLBUS. FINANCE must specify, along with the service itself, the mechanisms with which UTILSERV can change the resources allocated to the FINANCE service. It would be to the advantage of FINANCE to provide such mechanisms; UTILSERV would bill FINANCE only for the resources actually consumed by its service.

FINANCE will need to provision the identities of the SMALLBUS users who will consume the service. It will use these identities to control access and to bill SMALLBUS for consumption of the service. Any attributes of the service specific to the particular SMALLBUS customer that has been enrolled will be provisioned. The SLA between FINANCE and SMALLBUS will reflect the attributes of the agreement between FINANCE and UTILSERV.

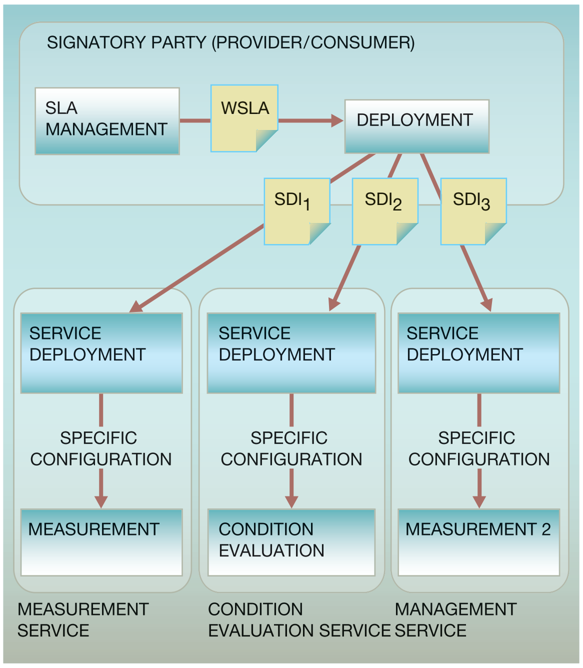
Another important component of an SLA management system is an arbitration engine, which is responsible for arbitrating between different elements of the SLA system when there is a shortage of resources. This arbitration engine is driven by the business goals of the service provider (these goals need to be updated as new subscribers are added).

Deployment of WSLA monitoring. The SLA associated with service delivery is specified using WSLA. The WSLA part of the contract specifies the QoS guarantees as agreed upon by provider and customer, the signatory parties to the contract. The components that use WSLA information as input, in particular those that are involved in monitoring contractual compliance such as the measurement service and the condition evaluation service, must be supplied with the appropriate information. The information to be supplied is based on:

- *Relevance:* Components should only have to deal with information that is relevant for them. Measurement services are affected only by measurement data and by interfaces to components they interact with. Likewise, condition evaluation services only need information concerning obligations. Furthermore, a party can use multiple services of both kinds in a distributed environment. The number of services used is not necessarily specified in the WSLA.
- *Information hiding:* Compliance monitoring may involve one or more additional parties (in addition to the signatory parties). Signatory parties do not need to share the entire SLA with the supporting parties. Signatory parties must analyze the SLA and extract relevant information for each supporting party. All parties need information on interfaces they must expose, as well as the interfaces to partners they interact with.

Thus, the deployment process contains two steps (see Figure 7). In the first step, the SLA deployment function of a signatory party generates and sends configuration information in the Service Deployment In-

Figure 7 Deployment of WSLA-compliance monitoring



formation (SDI) format to its supporting parties. In the second step, the service deployment functions of supporting parties configure their resources in order to perform their role in the process of SLA monitoring.

The syntax of the SDI format is similar to the syntax of the SLA language. Rather than containing a complete SLA, 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 responsible for and how to interact with other parties (either to obtain other metrics or to make them available in the form of SLA parameters). Information on obligations included in the SLA, or information identifying parties that the measurement service does not interact with, is not relevant for this measurement service. Similarly, condition evaluation services do not need to know how SLA parameters are defined by metrics. Thus, there is an SDI format for measurement services and one for condition evaluation services.

Certification and validation. Before a Web service is deployed, the service and the provisioning actions available for that service need to be certified and validated for use. Traditionally this validation would produce information needed for the capacity planning and forecasting tools that can be used to control the system in simple and subscription-driven provisioning. For the more sophisticated dynamic provisioning scenarios, additional information is required to enable SLA-driven service provisioning.

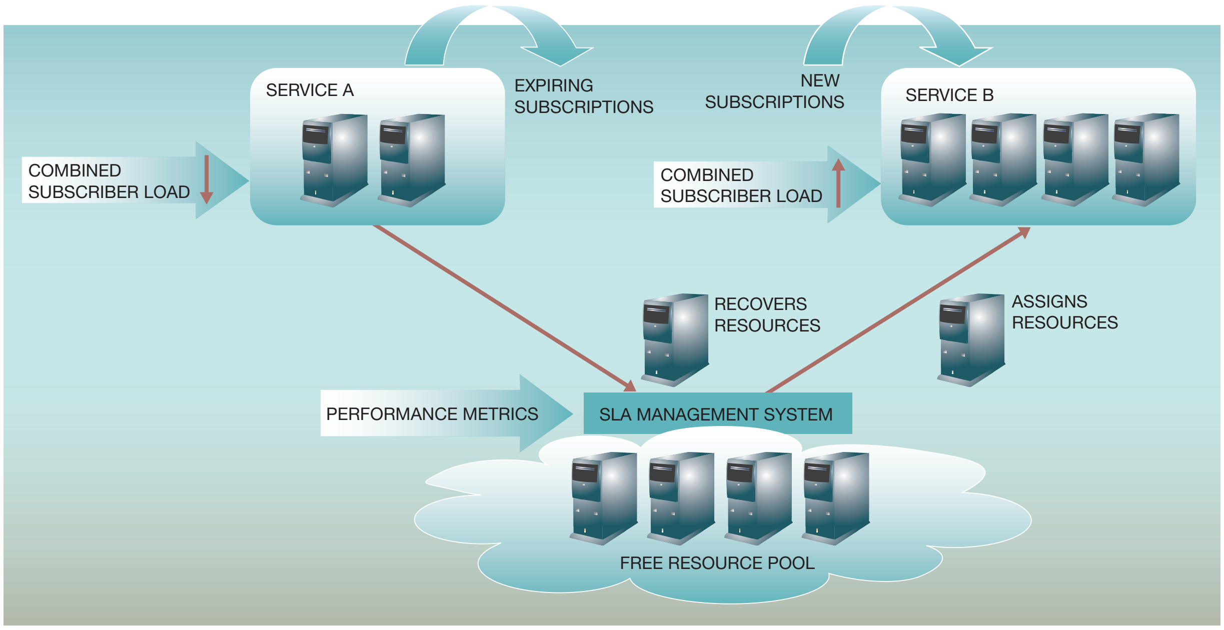
The information needed to dynamically provision a service can be determined by observing the behavior of the system under load. A load simulator can be used to apply various stress loads and measure changes in the service behavior. The resources dedicated to that service can also be varied, and the resulting changes in the service behavior measured. These measurements can be used to develop a profile of the service, and load testing can be used to further validate this profile.

When the service is deployed in the field, the SLA management system uses the service profile to optimize resource allocation to meet SLA commitments. This same profile can be used for capacity planning of the initial service deployment and to pre-deploy resources (to the service or free pool) in anticipation of subscriber growth or changing loads. The service profile may also be used as input to costing algorithms for fee-based hosting service providers. In our example, the behavior of the service offered by FINANCE would be certified. This would allow UTILSERV to dynamically adjust the resources allocated to that service to ensure a predictable response to changes made to meet the SLA that UTILSERV has with FINANCE.

The Web Service Stress Tools (WSST), available with the ETTK, can be used to load and stress test Web services. WSST offers a collection of stress-testing tools (of any local or remote Java method calls) for throughput and response time measurements. Tools currently include:

- Traffic generation engine with real-time control and comprehensive data collection and display
- Data review tool for display and analysis of collected data
- Support for parallel testing with multiple traffic classes
- Modular and extensible stress class behavior specified by experimenter using Java objects dynamically loaded at runtime
- Prepackaged set of classes that implement commonly useful behaviors

Figure 8 Dynamic provisioning



- Class generator tool for invoking SOAP (Simple Object Access Protocol) services through client-side stubs

The WSST is included in the Tools section of the Web Services and Grid "track" of the ETTK on IBM alphaWorks*[8]; the WSST is known there as the "Wide Spectrum Stress Tools."

It is interesting to note that the resources needed to host a service to be certified and the load testing system would make an ideal system for delivering on demand services.

Dynamic provisioning. During the normal operation of a service subjected to a consumer load, the SLA management system monitors the performance of this service and uses the monitoring information directly or via forecasting algorithms to predict changes in performance.[20] Referring to Figure 8, the SLA management system then dynamically provisions the service resources according to the business goals of the service provider. As illustrated in the figure, the combined load for service A is decreasing either due to expired subscriptions, anticipated loads given the time of day or season, or simply measured system load. The SLA management system removes a server from the pool used by service A, while still meeting the response time and throughput goals. In contrast, the combined load on service B is increasing either due to new subscriptions or measured or predicted temporal changes in activity. To maintain service B according to its SLA goals, new servers are added to the resource pool used to supply service B. This form of dynamic sharing reduces the overall cost of providing services while still meeting all SLA commitments. Performance metrics and objectives as defined in the SLAs are monitored by the SLA management system[20] and used to dynamically arbitrate assigned resource allocation.

To support dynamic provisioning, provisioning processes must be made available to the SLA system as actions or operations. These processes are similar to the processes used to provision the initial configuration for the service, though they may involve only parts of the entire provisioning flow, namely those specific to the service characteristics that are to be changed. As was seen earlier, the service offered by FINANCE would provide such processes. UTILSERV would leverage these processes to dynamically adjust the infrastructure resources allocated to the FINANCE service. This would allow UTILSERV to meet the SLA objectives of the service with the minimum resource cost. These savings would be passed on to FINANCE by charging only for the resources consumed in delivering the service. FINANCE thus minimizes its risk in offering a new service, as it would only be charged for the resources consumed by SMALLBUS, with UTILSERV automatically adjusting to increases and decrease in demand. This is a definite advantage over the former model of anticipating the worst-case load and requiring an up-front investment in the resources needed to meet that load while maintaining the SLA with SMALLBUS. Such resources would tend to be mostly idle during off-peak hours of consumption, and certainly they would be idle initially until the service was widely adopted by SMALLBUS subscribers.

The dynamic provisioning operations allow a service to be delivered in an autonomic fashion by a utility computing provider. An SLA-managed service will be able to consume resources in the most cost-effective manner, as well as react to changing resource availability and partial outages.[21] Careful planning can allow for resource overbooking by ensuring enough resources exist for maximum practical load rather than maximum theoretical load, using a free pool and dynamic provisioning to share these resources. By eliminating the need for manual provisioning, by controlling the provisioning of resources with an SLA management system, and by sharing resources, service delivery costs can be dramatically reduced.

Web-service execution environment

We now describe the Web-service execution environment and how SLA information is used in workload management, that is, managing response time and throughput of Web-service invocation based on customer identity.[22,23]

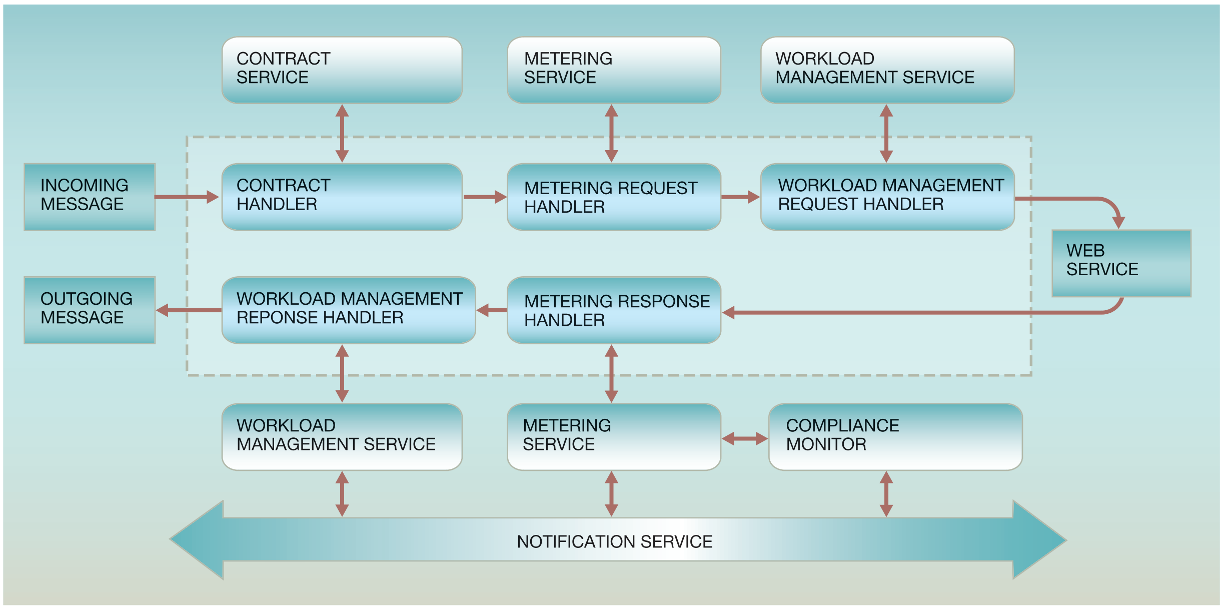
When integrating Web services into a business application, the SOAP engine should isolate the SOAP technology from the application developer as much as possible. This allows application developers to focus on their core task without having to worry about the auxiliary, transport-related duties managed by the SOAP engine. Similarly, the development and execution of the Web service itself should not be affected by any additional processing of the SOAP message. That is to say, while processing a SOAP message, it might be necessary for additional work to be done, either before or after the desired Web services are invoked. This additional processing, or provisioning, of Web services has been widely viewed as common enough that most SOAP processors have this ability built into them. They achieve this through the creation of a pluggable component in the SOAP engine known as a handler, a filter, or an interceptor. These components allow the administrator of the SOAP engine to easily decide what pre- or post-processing is required.

Of course this additional processing could be done many ways, and in particular it could be done within the Web service itself. However, by using handlers the exact customization required in a specific installation can be done without any code changes to the service. The Web Services community has already embraced this notion of pluggable pre- and post-processing components, and it has even been incorporated into the JAX-RPC (Java API for XML-based RPC) standard—the standard Java APIs for SOAP processing. Now that we have this notion of handlers, what can we do with them? One of the more natural uses of handlers is for processing that usually takes place outside of the realm of the specific work being done—for example, encryption. As in the HTTP (HyperText Transfer Protocol) world when SSL (Secure Socket Layer) is used, the applications on either end are usually not impacted by its use—it is all managed at the transport layer. Similarly, in the SOAP case we can use a handler on the client side to encrypt all, or part, of the SOAP message and a handler on the server side to decrypt it. Because this can be achieved easily through simple configuration modifications, no code changes should be necessary to the client or to the Web service itself.

How does all of this relate to the WSLA-driven automated management scenario? Consider that in the development of a Web-service-based application, it was realized there were some tasks that were common to all Web-service invocations. For example, all requests needed to be checked to make sure that the user invoking the service had a valid contract for this particular service. Also, additional processing is required for enforcing performance guarantees specified within that contract, and to track the amount of time or resources used during the processing of that request. These types of additional processing, while important to the complete application, should not be the concern of the specific Web service being invoked. These are, in essence, system-wide issues. Figure 9 illustrates specific types of handlers we would use in this scenario.[8]

Notice that we have quite a few handlers before and after the Web service itself is invoked. Before we go into the specific details about each handler and the corresponding service used by that handler, we first

Figure 9 Workload management functions implemented as common pluggable handlers



discuss how these handlers are designed and interact with one another. While handlers are independent pluggable components, this does not mean they cannot work together when necessary. For example, in our scenario the Contract Handler will determine the specific contract that the user has signed. The information about the contract will then need to be made available so that the other handlers can get access to it, if needed. To accomplish this, along with the SOAP message being passed from one handler to another, and ultimately to the Web service, the SOAP engine also passes some contextual information. The handlers then can use this storage to place data that components later in the flow can extract and use for their specific needs. There are also some additional components, the Compliance Monitor and the Notification Service, which will be discussed later. Finally, this configuration assumes that authentication has already been done by the application server and that the user's identity is made available to the system (as a unique key). Although how to authenticate a user is outside the scope of this discussion, it is still important to note that it must be done for a complete scenario to be developed. Next we discuss what each of these specific handlers will do.

Contract validation and contract detail extraction. The Contract Handler takes the user identity that has been added to the contextual information by the application server and uses Web-service contracting to check whether the user accessing the service has a valid contract in place at this time. In addition to this kind of authorization, in this scenario the Contract Handler puts more contextual information about this request's contract and its properties in the chain. Because of this, the other handlers do not need to access the information from Web-service contracting, which minimizes the overhead associated with such access.

Contract details, such as the contract identifier, information about the service and the operations under contract are put in, as well as the WSLA identification, throughput level, and throughput interval. All these details have been agreed upon in the contract.

Usage metering. The Metering Service, together with the respective handlers (Metering Request and Metering Response), allow for the metering of the Web service without requiring changes to the Web service implementation. It enables the collection of information related to the customer's service utilization. This utilization is expressed in terms of resource usage, or consumption. The collected information is defined in a *meter event*. The aggregation of these meter events forms the basis for charging and billing of customers. Each individual operation of the Web service that is requested by the customer is metered.

The consumption is measured by using different usage types. The type of service usage has been agreed upon earlier as part of the overall agreement in the usage contract. Four types are supported by the meter event:

- *Request-count-based:* This type counts requests to the service. It is also known as pay-as-you-go. A factor can be used to express multiple counts with one meter event. For example, a count of 3 may represent 15 requests. In the example, this meter event would be used for the number of stock transactions the customer would execute.
- *Time-based:* The type records the start and end time of the service request. A "start" meter event and a corresponding "end" meter event measure the duration. The meter events contain a special attribute that is used for correlating the corresponding events. In the example, this event would be used to detect response time violations.
- *Amount-based:* This type is represented by a pair of data, the actual value and a unit of measurement. With this type, amount consumptions can be collected. For instance, the pair (212000, KB) can be from the volume-based Internet connectivity service of the example. The consumption is metered in amounts of kilobytes of usage. The value is represented by using an integer data type to avoid any rounding inaccuracy.
- *Canceled:* This type is used to indicate that events with the same correlation identifier should be ignored. It is used for compensation activities.

The Metering Service has two receiving interfaces. One interface is able to receive single meter events, and the other can consume a batch of meter events. The later can be used for caching purposes. The Metering Service stores the passed events persistent to a database and adds time-stamp information to address audit requirements.

As shown in Figure 9, the Metering Request Handler receives the contextual data from the Contract Handler. Of special interest is the contract identifier that stands for the particular contract under which the service is requested. The Metering Request Handler creates meter events depending on the type of usage that was agreed upon in the contract, adds all required information including the contract identifier, creates a session identifier, and passes the events along the handler chain.

The Metering Response Handler gets the meter events from the context and creates new events, for example, the end time event for a start time event. Correlation identifiers (contract and session) are added, and the set of meter events is sent to the Metering Service.

For composite Web services, the meter event supports a parent session identifier to create tree-like structures of meter events.

Workload management. A service provider can offer each Web service in different SLA grades, with each grade defining a specific set of SLA parameters. Each grade is differentiated by SLO, base price, and performance penalty. The service provider uses a configuration tool to also create a set of traffic classes and map a <user, service, operation, grade> tuple into a specific traffic class. The service provider assigns a specific response time target to each traffic class. The workload management subsystem allocates resources to traffic classes and assumes that service operations assigned to a specific class have fairly similar execution times on a lightly loaded system.

The workload management subsystem controls the amount of resources allocated to each traffic class, thus controlling the response time behavior of the class.[24] In addition, the workload management subsystem is also responsible for policing the contracted throughput limit per customer.

Figure 10 illustrates the details of the workload management subsystem. The main components are: a workload management service, a global resource manager, and a management console. The workload management service may be responsible for coordinating the dispatching of requests to one or more back-end servers on which the target Web services are deployed. The server on which the workload management service itself executes may be one of the target servers.

The workload management service is composed of a queue manager, a scheduler, a load balancer, and a throughput policer. The throughput policer is responsible for dividing the input stream into a set of request flows and for monitoring the arrival rate of each request flow. A request flow is uniquely identified by a <contract Id, service, operation> tuple. Requests beyond the contracted maximum throughput limit are marked as out-of-profile and subsequently isolated and treated on a best-effort basis (in terms of meeting their response time objectives) by the queue manager. Thus, out-of-profile requests have minimal impact on the response time performance of customers conforming to their contracted throughput levels.

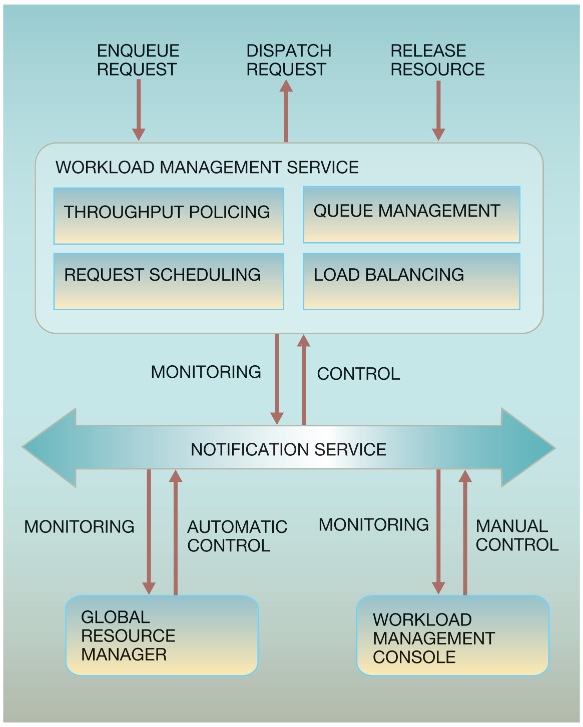
The queue manager implements a set of logical FIFO (first-in first-out) queues, one for each class. Upon receiving a classified request, the queue manager suspends the request and adds it to the logical queue corresponding to the request's class. A scheduler runs when a specific set of events occur and selects the next request to execute. The scheduler uses a weighted round robin scheme. We use a dynamic-boundary and work-conserving discipline that always selects a non-empty queue if there is at least one. After the scheduler selects a request, the queue manager signals the workload management request handler to resume the execution of the request. The queue manager collects statistics on arrival rates, execution rates, and queuing time and periodically broadcasts these data on the notification service.

When there are multiple back-end servers, the load balancer selects the server to which the request is dispatched. A simple round-robin load-balancing discipline is used, while ensuring that the number of outstanding requests dispatched to each server does not exceed the allocation assigned to it. When the execution of a request is completed, the workload management response handler reports to the queue manager the completion of the processing of the request. The queue manager uses this information to both keep an accurate count of the number of requests currently being executed and to measure performance data such as service time.

The Global Resource Manager (GRM) periodically adjusts scheduling weights and concurrency limits, taking into account current measurements of the offered load, server utilization, and server performance. The GRM periodically publishes the control settings via the notification service. The GRM allocates server resources dynamically in order to maximize the expected value of a given utility function in the face of fluctuating loads. The utility is a function of the performance delivered to the various classes.

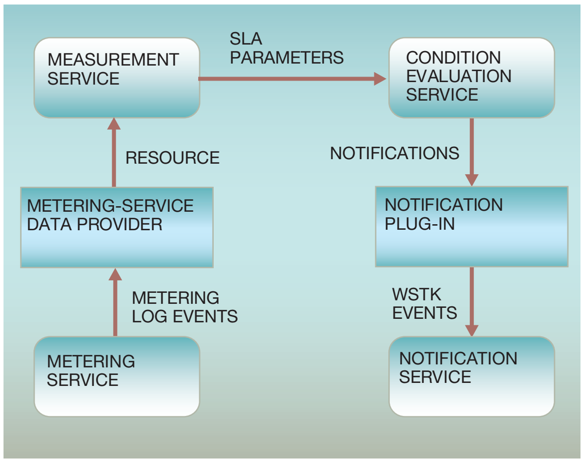
The Management Console provides an integrated GUI (graphical user interface) to the management system. It displays many of the monitoring and control data distributed over the notification service. It further allows "manual override" of GRM decisions. Finally, it displays and allows override of certain configuration parameters.

Figure 10 Workload management subsystem



Compliance monitoring. In the WSLA monitoring model, the flow of data through the Compliance Monitor (referred to in Figure 9) can be understood in terms of the components shown in Figure 11. The measurement service operates on basic inputs that it requests periodically from a data-provider plug-in, through a general-purpose interface (these basic inputs are known as resource metrics). It is the job of the plug-in to get the resource metrics, in this case by querying the Metering Service and extracting data from the results of that query. The measurement service aggregates resource metrics to higher-level SLA parameters on which the service level objectives are based. SLA parameters are sent to condition evaluation services that check the service level objectives based on those parameters. The condition evaluation service will then, as necessary, create notifications of violations, compliance, or other service states. These are passed through another defined in-

Figure 11 SLA monitoring model



terface to a plug-in that, in this specific case, knows that notices are sent in the form of ETTK events to the ETTK's notification service.

The measurement service and the condition evaluation service are general-purpose facilities for monitoring compliance of an arbitrary SLA, related to an arbitrary system. The SLA specification must therefore include the appropriate measurement service for a specific measurement, the appropriate condition evaluation service for a specific SLO, and the appropriate management service to which a specific notification must be sent. Then, the plug-in configuration data must indicate the plug-ins that get the resource metrics and the plug-ins that send notifications.

All of this comes together when a new SLA is deployed. First, the WSLA document for the SLA is split into one piece for the measurement service and another for the condition evaluator. In this specific context, where all of the compliance monitoring takes place within a single organization, such a splitting is unnecessary. However, the general WSLA framework takes into account the possibility that different organizations will handle different compliance monitoring subtasks, and there may be parts of the SLA that certain subcontractors are not privileged to see. Hence, the SLA information is split and deployed to appropriate components. The measurement service and the condition evaluator each then construct those internal objects necessary to perform the necessary monitoring. This includes use of the plug-in configuration data to help in the creation of appropriate plug-ins to gather resource metrics and to send notifications when guarantees are not met.

Summary and conclusions

Dynamic outsourcing of business processes and applications are crucial for businesses to be effective, efficient and flexible in meeting the requirements of fast changing market conditions. To be successful in dynamic outsourcing and to quickly form new business relationships depend on three critical factors: (1) open and emerging standards in accessing outsourced services, and in particular the use of Web services, (2) establishment of service agreements that include assurances on the quality of the service (SLAs), and business terms and conditions including pricing and penalties, and (3) automated management of the entire life cycle of business relationships, including: creation of the service offering, creation of SLAs with possible negotiation, provisioning of applications and environments, and monitoring of SLAs for both dynamic allocation of resources and compliance. Exploitation of economies of scale and automated SLA-based service management gives rise to utility computing systems that provide services in a cost-effective and efficient manner.

We have described in this paper the essential elements of a utility computing architecture for supporting the entire life cycle of an SLA and SLA-driven management of services. Within this conceptual framework, there are many design points in integrating various tools and technologies. A version of this architecture has been realized by integrating various technologies developed by teams from the IBM Research Division and the IBM Software Group. Work is continuing in the evolution and enhancement of existing technologies as well as in integrating new technologies within this framework, and in particular SLA negotiation and dynamic provisioning of resources.

Acknowledgments

This paper has drawn upon work jointly performed with many colleagues. In particular, we acknowledge the contributions of Paul Chen, Richard Franck, Joachim Hagmeier, Giovanni Pacifici, and Asser Tantawi to technologies that culminated in the management framework described in this paper. We thank the anonymous reviewers for their help in improving the presentation of this paper, and also a special thanks is in order to Asit Dan for his effort in coordinating this paper.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc.

Cited references

1. A. Dan, H. Ludwig, and G. Pacifici, *Web Service Differentiation with Service Level Agreements*, White Paper, IBM Corporation (March 2003), ftp://ftp.software.ibm.com/software/websphere/webservices/webserviceswithservicelevelsupport.pdf.
2. M. Kienzle, A. Dan, D. Sitaram, W. Tetzlaff, "The Effect of Video Server Topology on Contingency Capacity Requirements," *Proceedings of the Multimedia Computing and Networking Conference*, San Jose, Jan 1996, IS&T/SPIE (1996).
3. P. Grefen, H. Ludwig, A. Dan, and S. Angelov, *Web Service Support for Dynamic Business Process Outsourcing*, IBM Research Report RC22728, IBM T.J. Watson Research Center, Yorktown Heights, N.Y. 10598 (2003).
4. H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck, *Web Service Level Agreement (WSLA) Language Specification, Version 1.0*, IBM Corporation (January 2003), http://www.research.ibm.com/wsla.
5. K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02)*, Edinburgh, July 2002, *Lecture Notes In Computer Science*, Volume 2537, Springer-Verlag, Berlin (2002), pp. 153–183.
6. H. Gimpel, H. Ludwig, A. Dan, and R. Kearney, *PANDA: Specifying Policies for Automated Negotiations of Service Contract*, Research Report RC22844, IBM T.J. Watson Research Center, Yorktown Heights, N.Y. 10598 (2003).
7. K. Keahey and K. Motawi, *Taming of the Grid: Virtual Application Services*, Technical Memorandum ANL/MCS-TM-262, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439 (2003).
8. *Emerging Technologies Toolkit*, IBM alphaWorks Emerging Technologies, IBM Corporation, http://www.alphaworks.ibm.com/tech/ettk.
9. P. Bhoj, S. Singhal, and S. Chutani, "SLA Management in Federated Environments," *Proceedings of the Sixth IFIP/IEEE Symposium on Integrated Network Management (IM'99)*, IEEE, New York (1999), pp. 293–308.
10. S. Hepper and S. Hesmer, "Introducing the Portlet Specification," *JavaWorld* (2003), http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-portlet.html.
11. L. Lewis and P. Ray, "On the Migration from Enterprise Management to Integrated Service Level Management," *IEEE Network* **16**, No. 1, 8–14 (January 2002).
12. *Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language*, W3C Working Draft, World Wide Web Consortium (2003), http://www.w3.org/TR/wsdl12/.
13. *Business Process Execution Language for Web Services Version 1.1*, IBM Corporation, http://www.ibm.com/developerworks/webservices/library/ws-bpel/.
14. C. Overton, "On the Theory and Practice of Internet SLAs," *Journal of Computer Resource Measurement* **106,** 32–45, Computer Measurement Group (April 2002).
15. *Keynote—The Internet Performance Authority*, Keynote Systems, Inc., http://www.keynote.com.
16. Xaffire, Inc., http://www.xaffire.com/.
17. A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *Journal of Network and Systems Management, Special Issue on E-Business Management* **11**, No. 1 (March 2003).
18. *Autonomic Computing: Creating Self Managing Autonomic Systems*, IBM Corporation, http://www.ibm.com/autonomic/index.shtml.
19. D. Verma and S.B. Calo, *Service Level Driven Provisioning of Outsourced IT Systems*, Research Report RC22501, IBM T.J. Watson Research Center, Yorktown Heights, N.Y. 10598 (2002).
20. C. Crawford and A. Dan, "eModel: Addressing the Need for a Flexible Modeling Framework in Autonomic Computing," *Proceedings of the IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2002)*, IEEE, New York (2002), pp. 203–208
21. C. Ward, M. Buco, R. Chang, and L. Luan, "A Generic SLA Semantic Model for the Execution Management of e-Business Outsourcing Contracts," *Proceedings of the 3rd International Conference on e-Commerce (EC-Web 2002), Lecture Notes in Computer Science*, Volume 2455, Springer-Verlag, Berlin (2002), pp. 363–376.
22. G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef, *Performance Management for Web Services*, Research Report RC22676, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598 (2003).
23. R. Levy, J. Nagarajarao, G. Pacifici, M. Spreitzer, A. Tantawi, and A. Youssef, "Performance Management for Cluster Based Web Services," *Proceedings of 8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003)*, Colorado Springs, Colorado, March 2003, IEEE, New York (2003).
24. X. Zhu, J. Rolia, M. Arlitt, and A. Andrzejak, "Statistical Service Assurances for Applications in Utility Grid Environments," *Proceedings of the Tenth IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002)*, IEEE, New York (2002).

Asit Dan *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (asit@us.ibm.com).* Dr. Dan has been with the IBM Research Division since 1990 and is at the forefront of research and development in on demand computing and, before that, in transaction-processing architectures and video servers. He holds several top-rated patents in these areas and has received two IBM Outstanding Innovation Awards and eight IBM Invention Achievement Awards. Twice, he received the honor of IBM Master Inventor for his work in these areas. Currently, he is managing the Business-to-Business Integration Department that is working on the development of the infrastructure for supporting dynamic and SLA-driven Web services and grid and autonomic computing. Dr. Dan received a Ph.D. from the University of Massachusetts, Amherst. His doctoral dissertation received an Honorable Mention in the 1991 ACM Doctoral Dissertation Competition and was published by the MIT Press. He has published extensively, including several book chapters, and a book on multimedia servers.

Doug Davis *IBM Software Group, 3039 Cornwallis Road, Research Triangle Park, NC 27709 (dug@us.ibm.com).* Mr. Davis works in the Emerging Technology division of IBM as the technical lead for the IBM Emerging Technologies Toolkit. He was one of the original authors of Axis, the Apache SOAP engine, and his previous projects also include the WebSphere machine translation project, TeamConnection, and the FORTRAN 90 compiler. Doug has a B.S. degree from the University of California at Davis and an M.S. degree in computer science from Michigan State University.

Robert Kearney *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (firefly@us.ibm.com).* Mr. Kearney has been with IBM for 34 years—first in software development, and for the past 21 years, at the Watson Research Center. Formally educated in mathematics (University of Massachusetts, University of Wyoming, and Pennsylvania State University), his expertise is in operating systems and applications. He is currently interested in tools development, and especially in support of business-to-business applications.

Alexander Keller *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (alexk@us.ibm.com).* Dr. Keller is a research staff member in the Autonomic Computing Department at the Watson Research Center. He received his M.Sc. and Ph.D. degrees in computer science from Technische Universität München, Germany, in 1994 and 1998, respectively, and has published approximately 40 refereed papers in the area of distributed systems management. He joined the IBM Research Division in 1999. Dr. Keller's research interests revolve around change management for applications and services, information modeling for e-business systems, and SLAs. He is a member of the USENIX Association, the IEEE, and the DMTF CIM Applications Working Group.

Richard P. King *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (rpk@watson.ibm.com).* Mr. King joined the IBM General Systems Division in 1977 in Rochester, MN, where he worked on System/38™, especially in the area of system performance. He joined the IBM Research Division in 1981, where he is now a senior programmer. The projects he has worked on include fault-tolerant computing, the coupling of mainframe sysplexes, and high-performance intersystem messaging. Mr. King received a B.S. degree in industrial engineering and operations research from Cornell University in 1974 and an M.S. degree in operations research and industrial engineering from Northwestern University in 1975.

Dietmar Kuebler *IBM Software Group, Schoenaicher Strasse 220, 71032 Boeblingen, Germany (dkuebler@de.ibm.com).* Mr. Kuebler is a senior software engineer at the IBM Boeblingen Laboratory. Since joining IBM in 1990, he has held various positions in development, technical marketing, and project management, and has acquired experience in architecture and software development in multiple environments. His areas of expertise include object-oriented technologies, Java, WebSphere, and middleware technologies. Mr. Kuebler led the architecture and development of the Utility Web Services "Contracting, Metering and Accounting" for the Emerging Technologies ToolKit. Recently he has been involved in transferring these technologies into products and to customers. He is currently a member of the IBM On Demand Design Council (ODDC). He studied computer science at Stuttgart University, Germany, where he graduated in 1990.

Heiko Ludwig *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (hludwig@us.ibm.com).* Dr. Ludwig is a research staff member at the Watson Research Center, where he started as a visiting scientist in June 2001. As a member of the Distributed Systems and Services Department, he works in the field of electronic contracts and policies, primarily on WSLA. Previously he was a research staff member at the IBM Zurich Research Laboratory, where he worked on cross-organizational process management, service outsourcing, electronic contracts, outsourcing-related security aspects, and service modeling. From 1992 to 1996, he was a research and teaching staff member at the department of Office Automation at the Otto-Friedrich University in Bamberg, Germany. During that time he worked on co-operative planning and decision-making, and on the integration of workflow and collaborative applications. He holds a Master (Diplom) degree (1992) and a Ph.D. (1997) in computer science and business administration from Otto-Friedrich University. He published a book and several book chapters, various journal articles and conference papers, acted in program committees, and organized workshops in the area of computer-supported cooperative work, workflow management, e-business infrastructures, and contracts and policies.

Mike Polan *IBM Canada Lab, 8200 Warden Avenue, Markham, Ontario L6G 1C7, Canada (mpolan@ca.ibm.com).* Mike Polan is an architect in the Tivoli division of the IBM Software Group, working on the issues of provisioning and systems management in the on demand environment. His experience in software development includes microcode, computer communications, application development tools, and e-commerce systems.

Mike Spreitzer *IBM Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532 (mspreitz@us.ibm.com).* Dr. Spreitzer is a research staff member. He received his Ph.D. degree in 1989 from Stanford University. First at Xerox PARC, and later at the Watson Research Center, he did research work in programming languages and environments and distributed systems. His current focus is on performance management and performance characterization of clustered services.

Alaa Youssef *Computer Science Department, Alexandria University, Egypt.* As a research staff member at the Watson Research Center from August 1998 to August 2003, Dr Youssef was a member of the team that developed the first prototype of a quality of service and performance management system for Web services. Dr Youssef received his Ph.D. degree in computer science in 1998 from Old Dominion University, VA. He received his B.S. and M.S. degrees in computer science from Alexandria University, Egypt, in 1991 and 1994, respectively. His research interests include distributed systems, service management middleware, network and application-level quality of service, and security. Dr Youssef is a member of the IEEE.