Utility computing SLA management based upon business objectives

by M. J. Buco
R. N. Chang
L. Z. Luan
C. Ward
J. L. Wolf
P. S. Yu

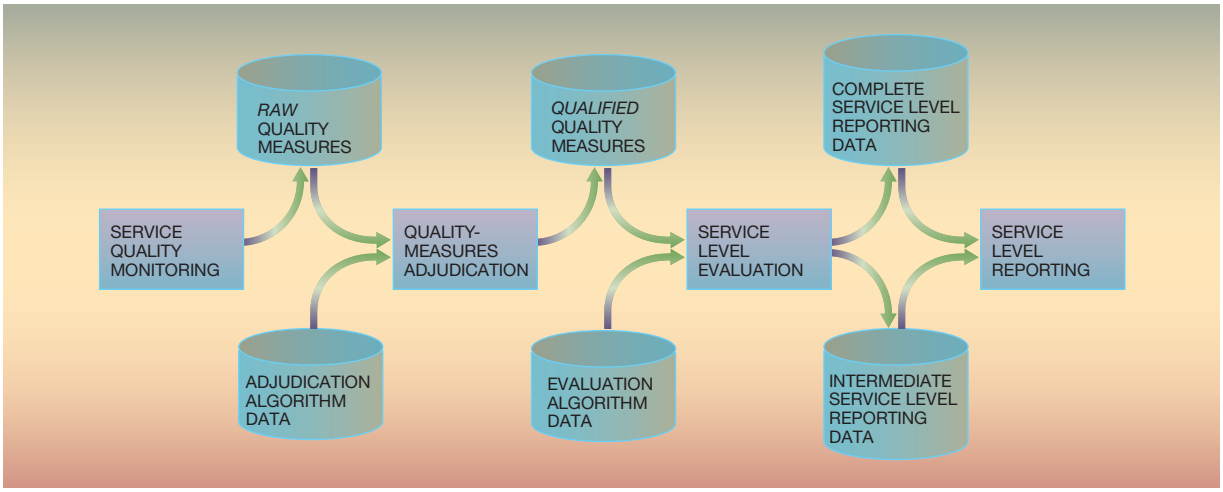
It has become increasingly desirable for companies worldwide to outsource their complex e-business infrastructure under the utility computing paradigm by means of service level agreements (SLAs). A successful utility computing provider must be able not only to satisfy its customers' demand for high service-quality standards, but also to fulfill its service-quality commitments based upon business objectives (e.g., cost-effectively minimizing the exposed business impact of service level violations). This paper presents the design rationale of a business-objectives-based utility computing SLA management system, called SAM, along with implementation experiences.

It has become increasingly desirable for companies worldwide to outsource their computing resources, e-business applications, and business processes, to focus on the growth of their core competency and to competitively improve their productivity by exploiting leading-edge computing technologies. Aiming at capitalizing on this information technology (IT) outsourcing trend, leading IT providers are exploring cost-effective means of maximizing the utilization of shareable computing and human resources under the utility computing model.[1] From the customer's viewpoint, the utility computing model promises on demand delivery of IT capabilities and cost-effective usage-based pricing schemes. Service-quality management objectives are assured by the provider in accordance with the established service level agreement (SLA) contract. The customer need not know the implementation details of the provider's service level management (SLM) processes.[2]

A *utility computing SLA* is an IT service contract that specifies the minimum expectations and obligations that exist between the provider and the customer of a utility computing service.[3,4] It includes one or more service level components, each of which specifies the measurement, evaluation, and reporting criteria for an agreed service-quality standard[5] such as:

- How *raw* quality measures (e.g., service availability or performance) for an agreed service component (e.g., on demand storage provisioning) in the contract will be gathered
- How *raw* quality measures will be adjudicated to become *qualified* quality measures (so that, for example, *"service outages caused by the customer or associated with contract maintenance provisions do not contribute to the total service downtime calculations"*[6])
- How *qualified* quality measures will be used to evaluate the achieved service levels (e.g., computing monthly Lotus Notes* availability as *"the monthly average availability of the Lotus Notes application running on the e-mail servers, weighted by the number of Lotus Notes IDs on each server"*)
- How service level evaluation results will be reported (e.g., *"monthly network latency statistics can be viewed at the following URL [uniform resource locator]"*)
- How unexpected disputes on service level evaluation results will be resolved

Figure 1 Business logic for service level reporting



Based upon the agreed set of quality standards (or service level targets), ramifications of not meeting or exceeding the standards can be explicitly included in the SLA contract. If a service level target (or a service level objective[2]) is linked with a penalty clause for a service level violation, it is considered to be a service level guarantee (SLG); otherwise, it is a service level intent. The clarity, attainability, and manageability of a service level guarantee are usually better than those of a service level intent in a commercial SLA contract.

A service level target in an SLA contract can be stated based upon objective quantitative measurement of computing system availability or performance (e.g., *"monthly availability of Individual Web Server will be no less than 99.7 percent"*) or business process efficiency or effectiveness (e.g., *"no less than 97 percent of on demand storage provisioning requests are fulfilled within two business days"*). The refund policies for missing service level targets can be specified relative to the service cost (e.g., *"credit customer one day of the service cost if the outsourced e-business infrastructure is unavailable more than 15 minutes a day"*) or in absolute terms (e.g., *"credit customer two thousand dollars if a monthly average network latency across the provider ISP [Internet Service Provider] access links to the ISP's backbone is higher than 95 milliseconds"*). A sample (abridged) Web hosting SLA contract is provided in the Appendix.

From the viewpoint of a utility computing provider, offering a few customer-neutral service functions atop a common service delivery infrastructure exploits economy of scale better than pursuing a high degree of customization of its service functions for every potential customer. This customer-neutral approach to establishing SLA contracts is adopted by most network and server collocation service providers.[7] However, a competitive IT outsourcing contract normally requires nontrivial customer-specific customization or extension of the provider's "standard" service offerings to accommodate the customer's unique IT outsourcing needs. When the number of such customer-specific SLA contracts grows, the complexity increases in the provider's service delivery environment and SLM processes.[8] A credible study on a leading IT service provider's SLA reporting cost, for example, has shown that several millions of dollars could be saved annually by reducing the cost of generating the monthly reports for 100 high-valued customer-specific SLA contracts by no more than 20 percent. Thus, it is important for a successful utility computing service provider to be able to satisfy its customers' demand for customer-oriented IT outsourcing functions with high-quality services and to fulfill all of its SLA commitments based upon business objectives (e.g., cost-effectively minimizing the exposed business impact of missing SLA commitments).

Figure 1 highlights the business logic for service level reporting (or SLA compliance reporting). It shows that the gathered *raw* quality measures must be adjudicated first before they can be used as *qualified* quality measurement. The service level evaluation

step can be triggered to generate the quality attainment reporting data for a past (completed) service level evaluation period or for the current evaluation period. After making changes to the input or implementation of any one of the steps, that step as well as the following steps must be re-executed to update the affected service level reports. Activation of such a report update process is necessary when, for example, the qualification status of a quality measure needs to be changed after a dispute about the quality measure is resolved between the customer and the provider.

In order to make timely adequate service management decisions based upon the provider's SLA commitments, the provider's SLA management system must be capable of performing on demand intermediate service level evaluations with support for adjudication processes for contractual quality measures. The intermediate service level evaluation results must be as accurate and current as possible so that appropriate SLM processes can be executed in a timely manner.

A leading utility computing provider must proactively: (1) maximize customer satisfaction with competitive service level reports (with regard to both coverage and attained quality), (2) minimize the exposed business impact of service level violations, and (3) lower the cost-to-quality ratio of executing SLM processes. However, these goals cannot be sufficiently and effectively supported by the existing service quality management products and common service management practices[9] for the following reasons:

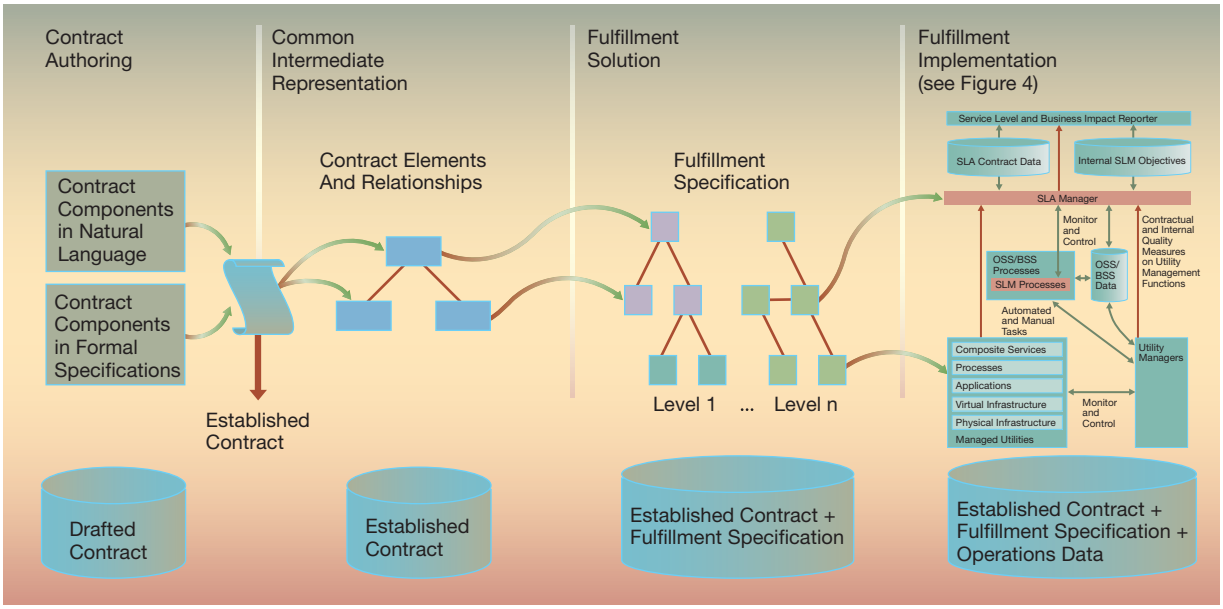
- Existing service quality management products do not support SLA compliance evaluations well because of their limited support of the adjudication processes for quality measures.[10]
- Contractual and internal quality measures on computing system health or performance are usually sent directly to service personnel (who usually manage systems by experience) or system management agents (which usually manage systems by infrequently changed thresholds or condition-action rules). Most service personnel and system management agents know little about the established SLA contracts; moreover, most of them incorrectly equate contractual service level targets to *raw* quality-monitoring thresholds.
- Service levels on efficiency or effectiveness of business processes (e.g., resource provisioning processes and problem resolution processes) are usually managed by a simple and static task prioritization scheme, such as those based upon severity levels.
- When computing-resource or human-resource contention situations, or both, are caused by unexpected system management alerts, ad hoc SLM processes are usually used to determine which management actions should be carried out first by the available service personnel or system management agents. Resolution-time-based business impact assessments of the alerts are not clearly linked with the provider's SLA commitments and the intermediate service level evaluation results for the affected SLA contracts.

Existing service-quality management technologies and methodologies, therefore, need to be improved to enable unified, business-oriented approaches to fulfilling SLA commitments.[11,12] This paper presents the design rationale of the utility computing SLA management system called *SAM* (SLA Action Manager) and our implementation experiences with it. The SAM project aims to develop a generic SLA management framework and an integrated set of advanced service level management technologies that among other benefits do the following:

1. Enable the provider to deploy an effective means of capturing and managing SLM-related contractual data as well as the provider's internal management data.
2. Enable the provider and the customer to review and analyze intermediate service level attainment reports on demand.
3. Assist service personnel and service management agents in ordering quality management alerts based upon the exposed business impact over time.
4. Automate the prioritization and execution management of SLM processes, including the assignment of SLM tasks to service personnel using continual optimization technologies.

The remainder of the paper is organized as follows. The next section presents the SLA semantic model of SAM with highlights on nontrivial data management needs for executing SLM processes based upon business objectives. The third section illustrates how SAM can be an integral component of a utility-computing operating environment. The fourth section presents the principal architectural components in SAM, and the fifth section elaborates on the service-quality alert ordering capabilities of SAM. In the sixth section we present the design of the process-based

Figure 2 Staged approach of SAM to capturing and managing SLA and SLM Data



SLM resource optimizing scheduler of SAM with performance comparison from event-driven simulations, and then we conclude our discussion.

The SLA semantic model of SAM

An SLA contract legalizes a mutual agreement between two parties on a service offering and an agreement on change management details. It also codifies how service quality will be objectively measured against agreed service level targets. Finally, it details which remedies are applicable to both parties when the targets are missed.

It is often unappreciated that related data for SLA compliance reporting (e.g., quality metric types, service level computation algorithms) are a proper subset of the SLM-related contractual data (e.g., pricing structure, SLA rebate computation algorithms) that are important for business-oriented SLA management. We refer to all SLM-related contractual data, which include the data that cannot or need not be explicitly part of the SLA document (e.g., unscheduled conditional service maintenance periods, recurring monthly service charges), as the *SLA Data.* We call the provider's service quality management data the *SLM Data.*

For example, a managed storage service contract that offers virtual disk space to a customer with an availability target requires that the provider manage the mapping between the virtual disk space and the corresponding physical storage resources. The SLA Data in this case comprise all contractually described data attributes associated with the availability of the virtual disk, including pricing, required capacity, availability target, and so forth. The corresponding set of SLM Data includes the data attributes associated with this mapping (e.g., physical storage server names, allocated capacity). Although the SLM Data and the SLA and SLM Data relationships must be managed well by the provider, such noncontractual implementation details do not have to be exposed to the customer.

To the best of our knowledge, there is still no commercially available satisfactory means of capturing and managing the SLA Data, the SLM Data, and the relationships between them in support of SLA-driven service-quality management approaches. The SLA and SLM data and relationships are often poorly managed as an ad hoc collection of paper documents, text files, spreadsheets, database records, and other items.

Figure 2 illustrates the staged approach of SAM to

capturing and managing SLA and SLM Data based upon contract life-cycle management activities. The SLA Data are established during the contract authoring phase and may be represented in a variety of formats (e.g., printed text, tables, arithmetic formulas, or formal language specifications). Once signed, the SLA Data are extracted and represented in a Common Intermediate Representation, independent of how the SLA contract is to be fulfilled. These data are then associated with a Fulfillment Solution, which is a multilevel implementation-independent relationship graph of the SLA-related SLM Data. For example, for a managed storage service contract, the SLA Data refer to the storage service components and associated service levels, whereas the SLM Data in the fulfillment solution refer to (at level 1) the set of attributes required to compute the contractual or internal service levels for the contract and link them (at level 2) to the set of fulfillment linkage specifications that locate the relevant data and algorithms. Exact details of the Fulfillment Solution are related to the nature of the contracts supported in SAM (i.e., the set of contract offerings represented in the Common Intermediate Representation) and could incorporate many linkage levels. Finally, the Fulfillment Implementation (e.g., service delivery environment configurations and other sources of relevant data) stores the actual SLM Data in various databases and locales as referenced by the Fulfillment Solution.

As part of our research effort in creating a generic and extensible intermediate representation of utility computing SLA contracts, we have developed a high-level information model that facilitates an iterative, top-down approach to capturing and managing the SLA Data based upon the purposes (or the semantics) of the needed contract data elements.[13,14] The model differs from other SLA-related information models[15,16] in that it focuses on the necessary and sufficient steps to affect business impact computation, a requisite step in proactive SLA management (including action execution and reporting). We have validated the SLA semantic model with more than 60 commercial IT outsourcing SLA contracts or templates, including those from the Information Technology Association of America (ITAA) Application Service Provider (ASP) SLA Library.[17] The set of SLA contracts or samples comprises more than 200 service level components in total for various service level targets for IT system availability and performance and business process effectiveness and efficiency. We believe the model is applicable to a very broad set of utility computing SLA contracts. The model can be extended by using other third-party models as its companion models.

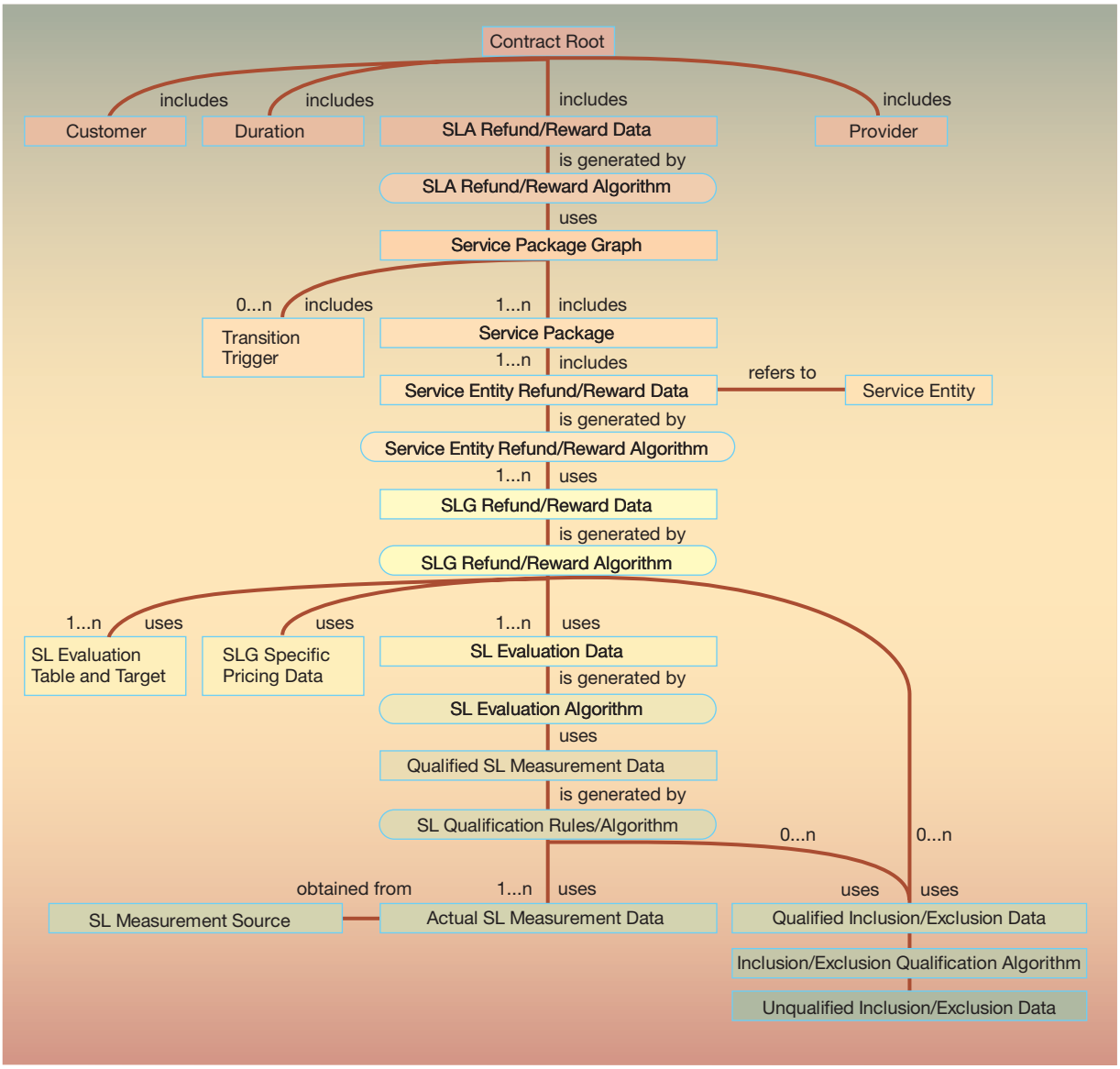
Figure 3 exemplifies the primary relationships between the core elements of the model in the Unified Modeling Language** (UML**)[18] with regard to the information needs for contract-wide SLA rebate reporting, an instance of SLA-based business impact reporting. The relationships have labels such as "use" and "includes." To facilitate top-to-bottom reading, some relationships are passive (e.g., "is generated by").

Starting from the top, we see in the diagram that an SLA contract includes information on the customer, which can be a single person, a single organization, or several organizations (e.g., the Joint Information Systems Committee members of the UKERNA contract[19]). The semantic element [Customer] captures, among other SLA Data, customer names, contact personnel, billing address, and so forth. [Customer] as well as [Provider] can be expanded through a third-party information model for person and organization, such as the person and organization model supported by the LDAP (Lightweight Directory Access Protocol) White Pages Schema,[20] Customer Profile Exchange,[21] and others. The element [Duration] captures contract start date, contract end date, contract termination terms and conditions, and other items. [SLA Refund/Reward Data] captures, for example, the contractual terms on the SLA rebate settlement interval (e.g., "*credits and premiums will be aggregated and settled on an annual basis*").

The relationships, represented in UML by solid lines, are an essential component in the semantic model and define specific interactions between the principal semantic elements. Although most of them are self-explanatory, several of them deserve special attention.

Figure 3 shows [Qualified SL Measurement Data] is generated by applying [SL Qualification Rules/Algorithm] to [Actual SL Measurement Data], i.e., *raw* quality measures. In computing service availability, for example, quality measures include "service up" and "service down" times. The qualification algorithm is specified in the contract and takes [Qualified Inclusion/Exclusion Data] as input (e.g., as customer-approved service downtime periods). Including this relationship in the SLA semantic model of SAM is important because IT outsourcing contracts often place restrictions on the total amount of service downtime for service maintenance per service level evaluation

Figure 3 Primary relationships between SLA semantic model elements of SAM for contract-wide SLA rebate computation



period. Service maintenance periods for a single service delivery site may also need to be adjudicated based upon the service maintenance periods of other service delivery sites (e.g., "*provider shall perform maintenance in a manner that ensures that one Web Hosting Environment with Geographically Diverse Web Hosting Environments will be fully operational and accessible at all times.*").

A single [SLG Refund/Reward Algorithm] may be linked with more than one [SL Evaluation Data] because the rebate computation for a single service level guarantee may use the evaluation results for several service levels. This nontrivial one-to-many relationship can be exemplified by the following excerpt from a commercial SLA contract: "*A Service Level Default occurs when Provider fails to meet a Min-*

imum Service Level during any month of the Term, at any time, or fails to meet an Expected Service Level with a Performance Category on four (4) or more occasions during a calendar twelve (12) month period following the Acceptance Date Plus five (5) months." We note that, in this example, the evaluation results for two types of service level targets are used in computing the yearly SLA rebate: monthly Minimum Service Level and yearly total number of occasions that the monthly Expected Service Level is missed.

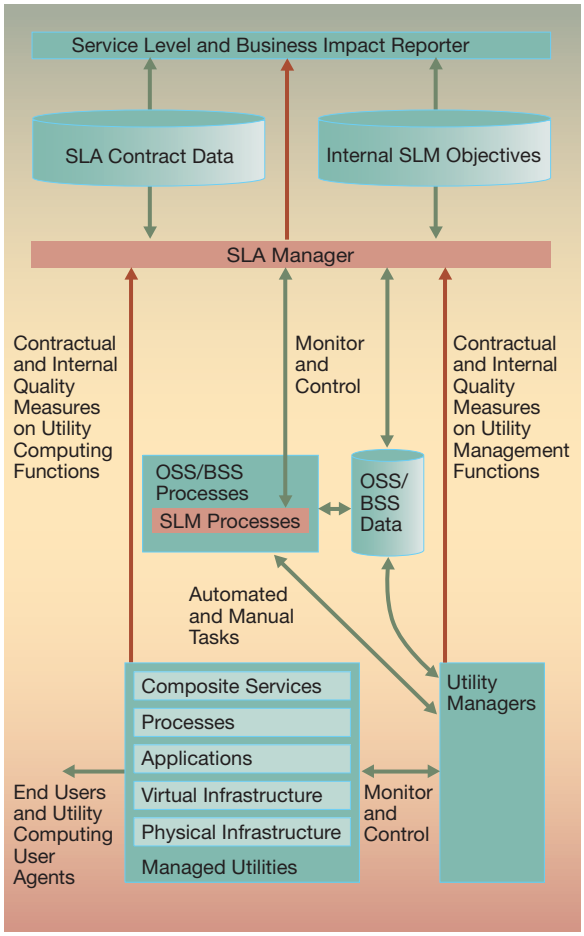
To compute the overall SLA rebate, the [SLA Refund/Reward Algorithm] uses an accumulation of individual service-level guarantee refund and reward data sets along with contract-wide SLA refund policy statements such as "*a maximum of 25 percent of the Customer's monthly fee will be credited each month (limit one credit per [network] line per month).*" The service level guarantee refund and reward data sets are organized into service entities, each of which captures the functional specifications for a service component in the contract, for example, help desk, network service, or learning utility service. We note that every service level component codifies the agreed quality for one and only one service entity, and several service level targets can be specified for a single service entity. We have seen an SLA contract with 12 service entities and 64 service level targets.[18]

The SLA semantic model of SAM includes a notion of service package graphs, composed of service packages and transition triggers. They enable the model to capture contractual agreements on how service level targets can change (e.g., "*customer may add or delete Performance Categories by sending written notice to Provider at least ninety (90) days prior to the date that such new Performance Categories are to be effective*") and how the SLA rebate computation algorithm can change (e.g., "*customer credit for the first month of a new order, which meets the [refund] requirements, is 25 percent of the prorated monthly fee*"). A [Service Package] can be considered as a data container for more than one [Service Entity] under contract-defined circumstances. The transition from one [Service Package] to another can be accomplished by some contract-defined event. It is possible that an intermediate [Service Package] is needed to make the service-level computations that are required during the transition neat and orderly.

SLA manager in a utility computing operating environment

Figure 4 illustrates, at a high level of abstraction, how an SLA manager (such as SAM) can be an integral component of a utility-computing operating environment that promises "*the delivery of standardized processes, applications, and infrastructure over the network, as a service, with both business and information technology functionality.*"[22] The utility computing services and the underlying physical IT infrastructure resources (including network routers, firewalls, server machines, and storage devices) collectively form the *managed utilities* in the environment. By leveraging existing utility computing functions such as virtual resources, a new service can be composed and managed as a composite service with characteristics that are distinguished from the sum of the constituent utilities. For example, an on demand application-renting service can be offered as a single utility

Figure 4 SLA manager in a utility computing operating environment



by integrating a set of software license-management utilities, license-aware application utilities, and data storage utilities.

In the advanced service delivery environment, each managed utility is associated with a *utility manager* that coordinates and automates the management of the physical and virtual resources of the utility through advanced IT system management technologies (such as policy-based resource provisioning and security management technologies).[23,24] A composite utility typically requires its own management interfaces and tools separate from those for its physical or virtual resources.[25] The aforementioned on demand application-renting service, for example, needs its own manager to manage the necessary integration modules for the service and to integrate the management functions of the constituent utilities.

Service-level and business-impact reporting data can be generated by the SLA manager based upon:

- SLA contract data
- Provider's internal service level management objectives
- Operations support system/business support system (OSS/BSS) data
- Contractual and internal quality measures on utility computing functions (e.g., availability and performance of the managed utilities)
- Contractual and internal quality measures on utility computing management functions (e.g., effectiveness and efficiency of the OSS/BSS processes).

Besides the SLA-related monitoring and reporting functions, the SLA manager can perform SLA-related control functions as well. It can leverage state-of-the-art IT system management technologies (e.g., continual optimization, policy-based management, autonomic computing, and grid computing) as the utility managers do.

The SLA manager and the SLM processes that it manages actually provide a process-based control to the utility manager interfaces and to the utility management interfaces. The SLM processes can be manual, semiautomated, or fully automated. The result of executing either an automated or a manual service-quality management task is usually a modification to the overall configuration of the managed utilities. The overall OSS/BSS process instances essentially orchestrate all of the utility computing management activities in the SLA manager, utility managers, and managed utilities.

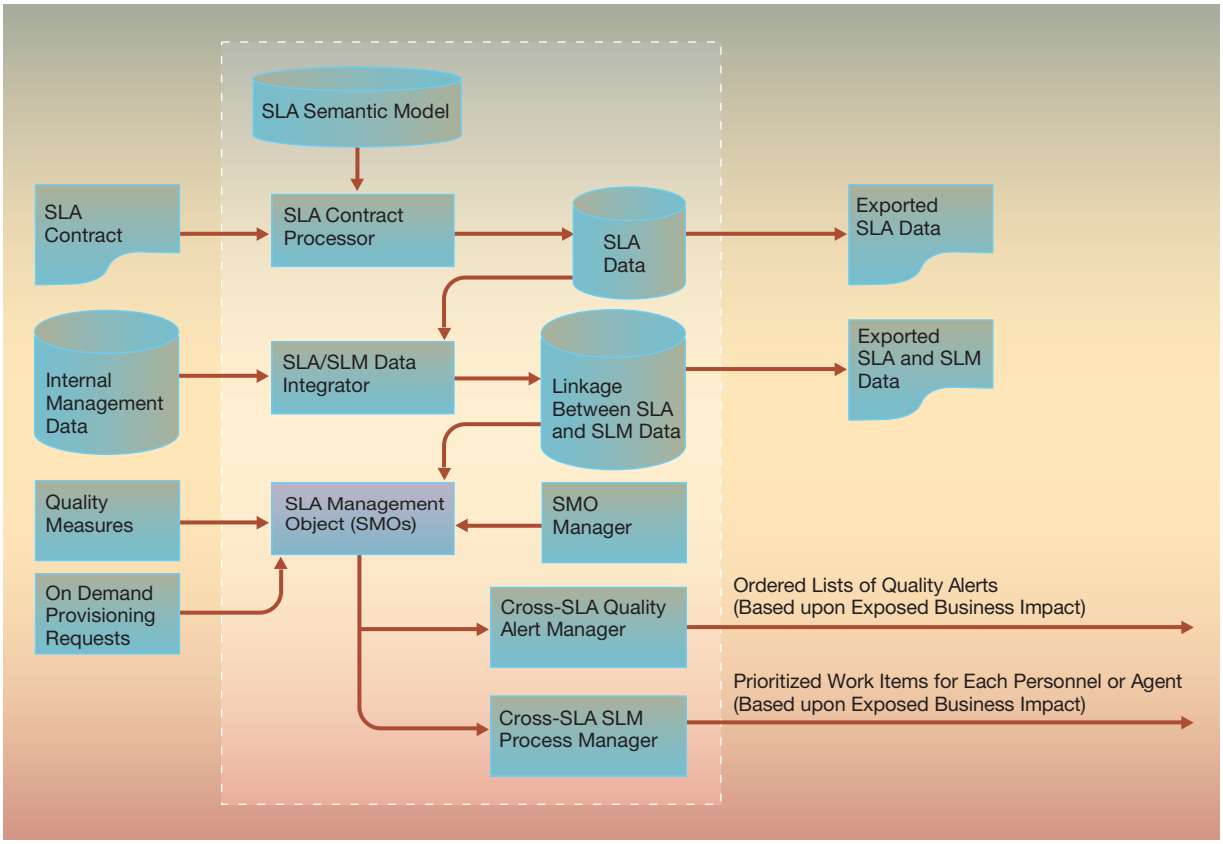
Principal architecture components in SAM

The SAM system is designed to be an extensible distributed SLA management environment that enables a utility computing provider to deploy an SLA-driven, business-impact-based SLM process management solution that not only can assist service personnel and service management agents in ordering quality management alerts based upon exposed business impact, but also automate the prioritization and execution management of SLM processes, including assigning SLM tasks to service personnel and service management agents.

Figure 5 illustrates the primary interactions between the principal components in SAM. After an SLA contract is established, the SLA Data are imported into SLA Data storage via an SLA contract processor based upon the SLA semantic model of SAM (see Figure 3). The SLA Data are then linked with the necessary internal SLM Data via an extensible SLM data management framework (the SLA and SLM Data Integrator) based upon the staged approach of SAM to capturing and managing SLA and SLM Data (see Figure 2). Every contract is bound with a unique SLA management object (SMO) in SAM, which interacts with the other service management components to manage the execution of the contract (e.g., processing contract-specific on demand resource-provisioning requests).

We note that accessing the SLA and SLM Data of SAM through a data management system using data access interfaces based on XML (eXtensible Markup Language) allows the implementation details of the SLA and SLM Data storage design to be hidden from other SAM components and related applications. This is particularly effective for SAM because of the difficulties inherent in describing a universal SLM Data schema for all contract offerings before their definition. A relational table-based data storage must either represent parameters as a set of generic tables (e.g., ParameterTable, ParameterRelationTable, and ParameterTypeTable) or represent parameter sets as text-based attributes within other entities. Both approaches introduce a weak notion of semantic integrity by obscuring the essential data relationships. Semistructured data storage holds the promise of greater flexibility, but it is still somewhat experimental. An XML-based data management interface permits both.

Figure 5 Key component technologies in SAM



Every established SLA contract is associated with one and only one SMO in SAM. Each SMO essentially transforms the associated SLA contract into an active computing entity in the distributed SLA management environment. Common functions supported by each SMO include, among others:

- Managing the SLA and SLM Data for the associated contract
- Performing complete and intermediate SLA compliance evaluations according to the business logic for service level reporting (see Figure 1) with support for both manual and automatic adjudication processes of quality measures
- Associating anticipated quality alerts with contract-specific (exposed) penalty functions
- Driving the determination of one or more SLM processes for each quality alert specific to its contract (by working with service personnel and other service management agents)
- Associating the needed SLM processes for handling a specific quality alert with contract-specific (exposed) penalty functions

Implementation details of the SMOs are hidden from the rest of the system via a common XML application programming interface (API) to the SMOs that is based on SOAP (Simple Object Access Protocol). With the recognition that every commercial SLA contract includes a clause for resolution of disputes, every SMO is required to re-evaluate complete and intermediate service levels and related business impact assessment data when the qualification status of a quality measure recorded in SAM is changed. The cost and efficiency of managing the life cycle of utility computing SLA contracts can be improved significantly via the SMOs and the SMO manager, which supports mainly contract change management activities.

The Cross-SLA Quality Alert Manager gathers quality alert data from all of the SMOs, normalizes and aggregates the penalty functions across all contracts, and maintains one or more ordered lists of (active) quality alerts in terms of business impact. The ordered lists of quality alerts can be formatted in XML or HTML (Hypertext Markup Language) and distributed to other service management agents and personnel in support of their decision-making needs. The lists are updated when the status of a quality alert changes.

The Cross-SLA SLM Process Manager gathers SLM process execution management requests from all of the SMOs, normalizes and aggregates the penalty functions across all contracts, and continually optimizes the execution of the SLM process instances based upon the provider's business objectives (or SLM process execution policies). The SLM processes are prioritized not only according to the penalty functions, but also according to the needed resources (e.g., qualified service personnel for a manual task) for each step of the processes when the resource requirements are available. The SLM processes can be implemented by means of several workflow engines.

Service-quality alert ordering in SAM

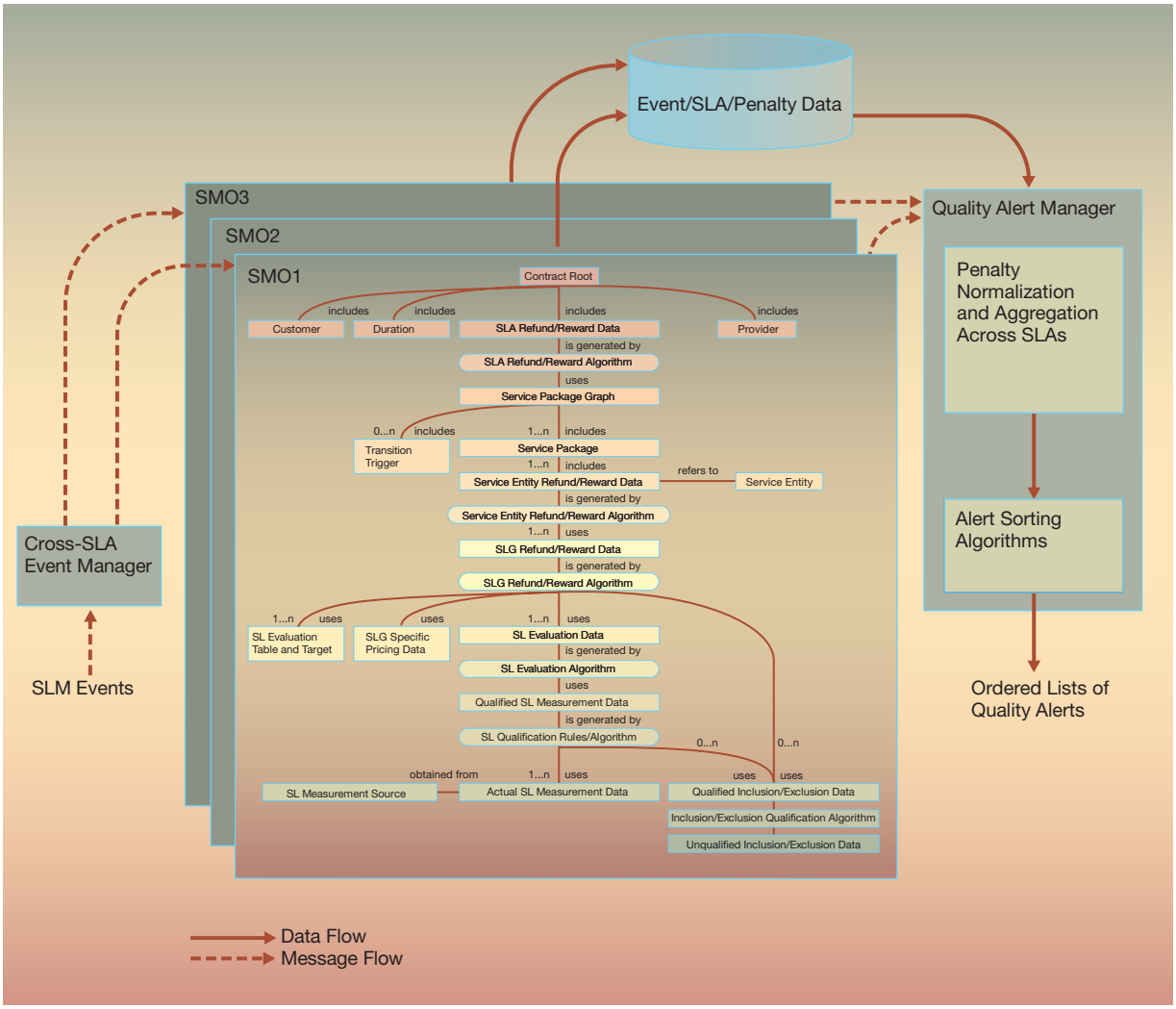
The working principles of the approach used in SAM to ordering cross-SLA quality alerts by exposed business impact over time (or by resolution-time-based penalty functions) are illustrated in Figure 6. It shows the quality measures that may affect multiple contracts are first received by the Cross-SLA Event Manager of the system via SLM events, each of which conveys a condition at a particular time (e.g., "server X is down at time T," "server X is up at time T," and "new resource provisioning request X arrives at time T") that deserves the provider's attention according to all of the contract-specified quality measurement criteria and service level targets. When receiving a contract-specific SLM event from the Cross-SLA Event Manager, the SMO processes the event according to its own SLA and SLM Data. If a new quality alert must be generated, the SMO assesses the exposed business impact of the alert on the basis of the contractual terms and latest quality measurement statistics for the current service level evaluation interval, and associates the alert with a penalty function in accordance with the time the alerted condition will be removed. The penalty functions for the alert from all of the affected SMOs are normalized and aggregated in the Cross-SLA Quality Alert Manager. The manager ensures that the ordered lists of cross-SLA alerts which it maintains are current and makes the lists available to its client applications, such as service personnel's Web browsers and service management agents.

We note that the quality-alert-ordering capability of SAM is developed mainly as a decision support function. Each supported alert-ordering algorithm reflects a distinct preference of the receivers of the alert lists. Figure 7 exemplifies three penalty-function-based alert-ordering algorithms that we have implemented in our SAM prototype, assuming the business impact metric in use is the additional (exposed) SLA rebate amount. The penalty diagram in the figure shows that there were two action-demanding quality alerts when the diagram was generated. It also shows that the provider was exposed to an additional cumulative penalty of $200.00 if the alert "S1 down" was not resolved within 15 minutes (relative to the time that the diagram was generated) and $600.00 within 45 minutes. Similarly, the provider was exposed to an additional cumulative penalty of $50.00 if it did not resolve "S2 down" within 10 minutes, $100.00 within 20 minutes, and $150.00 within 40 minutes. Note that the penalty amount data shown in the figure are used to facilitate the presentation. Service personnel, for example, need not see the data in practice.

The Earliest Higher Penalty Time–Complete Order (EHPT-CO) algorithm sorts all of the exposed (or predicted) penalty points in chronological order on the basis of their respective penalty time. The Earliest Higher Penalty Time–Partial Order (EHPT-PO) algorithm takes into account only the earliest penalty point of each SLM event. Instead of sorting the penalty points by their respective penalty times, the Higher Penalty Rate First–Partial Order (HPRF-PO) algorithm sorts the earliest penalty point of each quality alert on the basis of the ratio of extra penalty amount and time to the penalty point. Notice that "S1 down" is placed before "S2 down" by this algorithm.

Although the output generated by EHPT-PO or HPRF-PO appears more concise and user-friendly than that of EHPT-CO, EHPT-CO output facilitates exploiting estimated resolution times for action-demanding SLM events. For example, if it would take more than 10 minutes to complete the needed SLM process for "S2 down" and less than 15 minutes for "S1 down," the provider should execute the SLM process for "S1 down" first to minimize the total exposed penalty.

Figure 6 Ordering cross-SLA quality alerts by exposed business impact

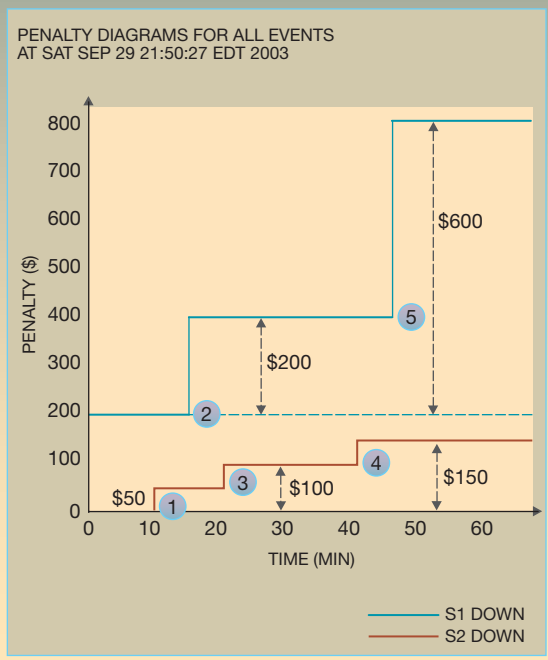


SLM process execution management in SAM

The most significant aspects of the industrial revolution at the beginning of the Twentieth Century were the innovations led by the automation of production. Most contemporary service providers understand the importance of achieving revenue and profile goals by integrating and automating OSS/BSS processes and are seriously reviewing and improving the execution of their business processes.[26] In light of the complexity of implementing and managing high-quality utility computing services, it is important for a utility computing provider to take advantage of contemporary process automation and integration products (such as WebSphere* Process Manager and WebSphere InterChange Server) and make the execution of its SLM processes cost-effective, flexible, and manageable.[27]

The Cross-SLA Process Execution Manager of SAM automates the prioritization and management of discrete service management tasks at the SLM process level according to the resource needs of each task by means of continual optimization technologies. It enables SAM to be an autonomic SLM process execution management system in which monitoring,

Figure 7 Examples of penalty function based quality alert ordering algorithms



EARLIEST HIGHER PENALTY TIME - COMPLETE ORDER
AT SAT SEP 29 21:50:27 EDT 2003

	RANK	EVENT	TIME TO NEXT PENALTY	ADDITIONAL PENALTY
1	1	S2 DOWN	10	$50
2	2	S1 DOWN	15	$200
3	3	S2 DOWN	20	$100
4	4	S2 DOWN	40	$150
5	5	S1 DOWN	45	$600

EARLIEST HIGHER PENALTY TIME - PARTIAL ORDER
AT SAT SEP 29 21:50:27 EDT 2003

RANK	EVENT	TIME TO NEXT PENALTY	ADDITIONAL PENALTY
1	S2 DOWN	10	$50
2	S1 DOWN	15	$200

EARLIEST HIGHER PENALTY RATE - PARTIAL ORDER
AT SAT SEP 29 21:50:27 EDT 2003

RANK	EVENT	TIME TO NEXT PENALTY	ADDITIONAL PENALTY
1	S1 DOWN	15	$200
2	S2 DOWN	10	$50

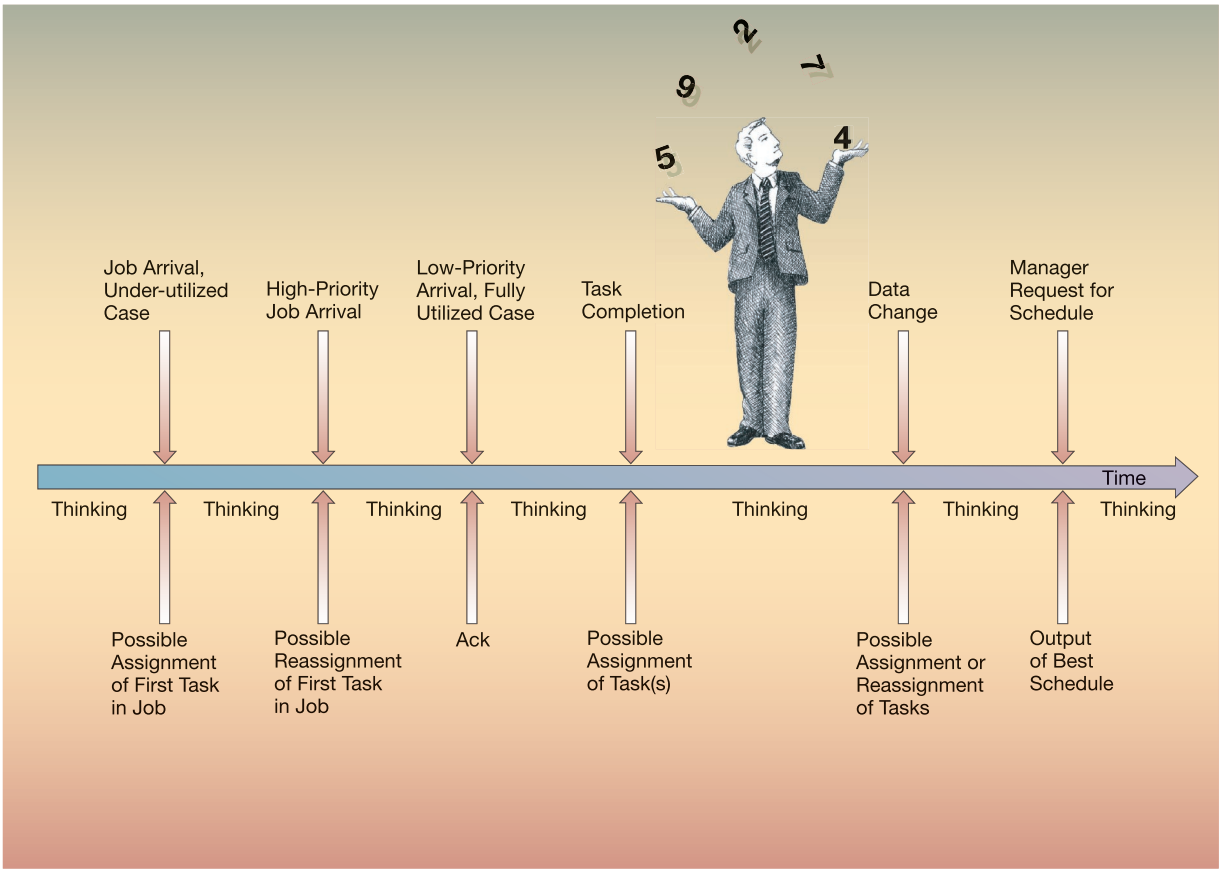
analysis, planning, and execution of SLM processes can be done automatically in accordance with business objectives.[28] This process-centric self-managing approach to fulfilling utility computing SLA commitments enables the service delivery team to focus on managing SLM processes in business terms, rather than resolving discrete system availability or performance issues by intuition.

We have developed an optimizing scheduler for the process execution manager so that SAM can manage the task assignments of service personnel and service management agents well. Note that the use of the word *optimizing* here refers to the goal rather than to the result, in much the same spirit as the expressions optimizing compiler and query optimizer. For technical reasons, as will be outlined below, it is extremely unlikely that the exact optimal solutions found for most problem instances considered by the SAM scheduler could be found in an acceptable amount of time.

The SAM scheduler assigns SLM tasks to service personnel (or employees) and software agents on the basis of a job-based task management model. The tasks in each job are assumed to be performed according to chain precedence; that is, there is a strict sequencing of the tasks within a job, and a given task can only be started when the previous task is complete. Every job and task can be associated with a penalty function. The scheduler tries to minimize the sum of the exposed business impact, one summand for each job or task, based upon its absolute or relative completion times.

The SAM scheduler takes into account personnel availability information, including, for example, lunches, breaks, and shift start and end times. Personnel skills and training on various tasks are also considered. Essential tasks that cause built-in delays but do not directly involve the existing personnel can be modeled. Such delaying tasks might pertain to work performed by software agents, work performed by personnel from other shops, and similar work situations. The scheduler can allow the common assignment of tasks to personnel, so that an employee assigned to one task would also be assigned to all other tasks in a collection of tasks. It also allows a set of tasks to be defined in a way that they cannot be handled simultaneously, perhaps as a result of common resource requirements. Multiple priority levels of jobs are modeled. If a higher-priority job arrives, it will cause immediate preemption of a task

Figure 8 Sample scheduler time line



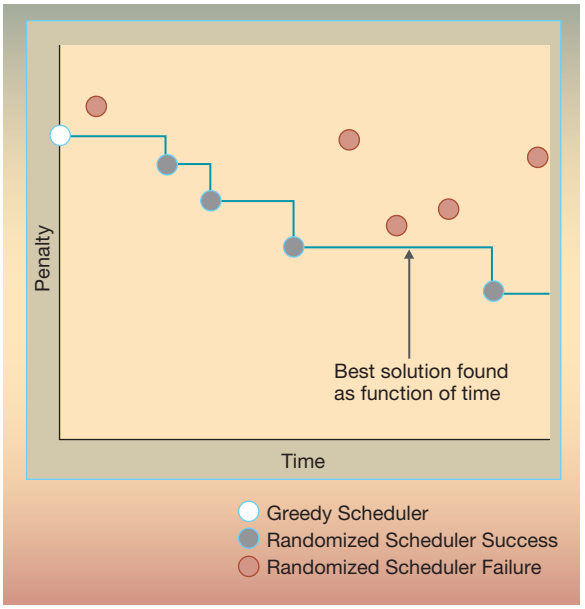
associated with a lower-priority job, as long as that preemption is useful.

The summands in the function to be optimized are generalizations of the sort of deadline penalties typically studied in scheduling theory.[29–31] The scheduling problem of assigning tasks to employees and other resources while minimizing the total penalties belongs to a class of so-called NP (nondeterministic polynomial) hard mathematical problems,[32] for which exact solutions are essentially intractable to attain. Despite the fact that a special case of our scheduling problem is shown to be NP hard in Blazewicz et al.,[29] we observe that the potential savings in penalties possible with a good quality scheduling tool can be quite dramatic as compared with an ad hoc solution. Moreover, we think customers will be significantly more satisfied if such a scheduling tool is implemented, because the assignments of tasks in jobs to employees and other resources will be fairer. The result will be greater customer loyalty, an intangible but very real benefit.

The SAM scheduler is event-driven, responding synchronously to each event arrival with a (possibly empty) list of new preemptive or nonpreemptive assignments, or both, followed by an acknowledgment that the list of assignments has been completed. If there is no work for the scheduler to do, which occurs when there are no queued tasks to consider, it simply "sleeps," waiting to be awakened, for example, by a subsequent job arrival. If any queued tasks exist, it "thinks about," or analyzes, possible variations of the resulting scheduling problem instances.

Figure 8 illustrates a sample scenario in which the SAM scheduler synchronously responds to stochastically arriving events such as job arrivals, task com-

Figure 9 Greedy versus randomized heuristic scheduler



pletions, data changes, and manager requests for a schedule. Time is represented on the horizontal axis. Each event (the top part of the figure) may cause the SAM scheduler to return scheduling decisions and the like (the bottom part of the figure). Then the SAM scheduler goes back to thinking, as illustrated by the number juggler, until the next event occurs.

Because the scheduler must not be a bottleneck itself, it must decide quickly about a problem instance that it has not necessarily anticipated. For example, new job arrivals generally cannot be anticipated with much accuracy, and the scheduler therefore does not attempt to analyze such problem instances in advance. But an arriving job may be of high priority, and thus the first task in that job might preempt an existing task assignment. Or a job might arrive when appropriate personnel are idle. This quick response philosophy argues for a scheduling scheme that is fast and produces a solution that is of at least reasonably good quality. More specifically, its immediate task assignments and reassignments should not be so poor that decent quality continuations of the schedule become impossible. The scheduler employs a greedy scheduling heuristic for this purpose. Although we do not have sufficient space to describe this scheme here, we note that it is very similar to the scheduling algorithm described in Reference 33 for an apparently quite different problem. A complete description of the greedy algorithm used for the problem in this paper is given in Reference 34. This greedy scheme is employed to initiate analysis of task completion events as well.

In contrast, there should also be a gracefully interruptible scheduling scheme which, given sufficient computation time, can potentially improve upon the solution quality obtained by the greedy algorithm. Thus, the scheduler employs a randomized scheduling heuristic as well. It is gracefully interruptible in the sense that each randomized schedule can be produced quickly, so that stopping the scheduler immediately will only disrupt one of many iterations of the algorithm. The best solution found is always kept. For details on randomized algorithms see Reference 35. A complete description of the randomized algorithm used for the problem in this paper is given in Reference 34.

Figure 9 illustrates a possible sequence of schedules computed for a particular potential problem instance. The x-axis represents computation time, and the y-axis represents the total exposed penalty. From left to right, the first circle, shown as white, refers to the greedy heuristic scheduler, which always runs first. The remaining orange and gray circles refer to multiple instances of the randomized scheduler. In particular, an orange circle indicates that no improvement has been achieved during that scheduler iteration. The corresponding solution is ignored. A gray circle indicates that the solution achieved during that scheduler iteration represents an improvement and is thus the current best solution. The corresponding solution is retained until such time as it is displaced by a better one.

The scheduler will never be idle in the presence of queued tasks, that is, if any scheduling problem remains to be analyzed. Moreover, it must consider an essentially infinite collection of problem variants. It is not known precisely how long each task will take to be completed, and these completion time variations can impact the quality of the scheduling subtly but considerably. Therefore, the scheduler tries to consider a set of problem variants that at least approximates all of these possibilities, essentially by betting both on which task will complete first and approximately when that task will be completed. The scheduler also tries to hedge these bets by thinking about all of them. At the beginning of each period of computation, the scheduler quickly solves a network flow resource allocation problem.[36] The goal

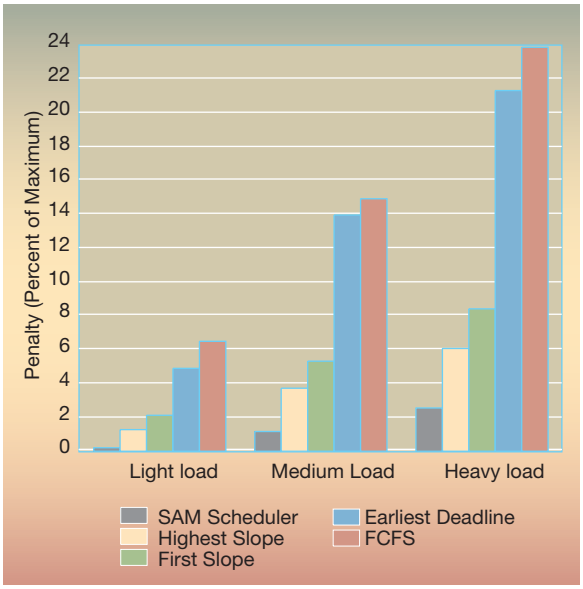
is to find the optimal "think time" allocations. This network flow covers all the possible task completion time bets, maximizing the weighted average of the expected improvement as a function of time, thus minimizing the expected penalties paid. The network flow formulation also ensures that the scheduler will have thought about, or analyzed, the actual first task to be completed *by the time it actually does.* Complete details on this network flow problem are given in Reference 34.

It might be useful to consider an analogy here to the scenario of a chess-playing computer playing a human challenger. The computer would typically think, or analyze, continually, trying to optimize its next move, regardless of whose turn it is. When it is the human's turn, in particular, an aggressive chess-playing computer might think about which move the human might make, and when. The computer could then allocate an appropriate amount of time for each of these possible "bets," and thus have an answer ready instantly whenever the human makes a move.

In light of the dynamic changing nature of the service delivery environment, the SAM scheduler only reveals the very next, immediate, task assignment to an employee (upon completion of the previous task), the next preemptive task reassignment (upon the arrival of a higher-priority job), and so on. Such a just-in-time philosophy has the benefit of not showing an employee any hypothetical, in-the-future task assignments, so that the employee is not confused if reassignments are made because of arrivals and data changes beyond the ability of the scheduler to anticipate. Additionally, the employee cannot pick and choose among several tasks, perhaps making a non-optimal choice. The employee sees one task at a time.

Figure 10 shows the relative merits of the job and task scheduling algorithm of the SAM scheduler and several other common task-scheduling heuristics. The "First-Come-First-Serve (FCFS)" algorithm simply schedules ready tasks in order of job arrival. Roughly speaking, the "Earliest Deadline" algorithm first schedules the ready task that has the earliest deadline. Similarly, the "First Slope" algorithm first schedules the ready task for which the first ratio of incremental penalty to incremental time to deadline is largest. The "Highest Slope" algorithm first schedules the ready task for which the highest ratio of incremental penalty to incremental time to deadline is largest. The SAM scheduler does the best here, in part because it exploits the most information and thinks more continually than with the other algorithms. We note that the SAM scheduler is modular, and its continual optimization framework is applicable to other application domains.

Figure 10 An event-driven simulation-based evaluation of the optimizing scheduler of SAM



The figure is generated based upon a set of event-driven simulation experiments. The simulation model assumes a Poisson arrival of jobs (low:med:high = 1:2:3) with low arrival rate chosen to achieve the threshold penalties, five job types (normal distribution), 10 employees, and an average of three penalty points per job. Additional details and further experimental results are provided in Reference 34.

We note that, in practice, an SLM process may be far more complex than a chain precedence task list. A workflow engine generally supports the modeling of an SLM process as a network of tasks represented by a directed graph. The order in which tasks are performed is determined by control flow links between tasks. These links can be conditional (that is, resolved at run time). A task can be the starting point or ending point of any number of links (allowing for parallel execution). Moreover, a task can be a single entity (e.g., a program), a subprocess (allowing for simplification and reuse), or a block of tasks that are repeated until some condition is met (allowing looping behavior).

Figure 11 A sample on demand storage-provisioning process captured as a workflow

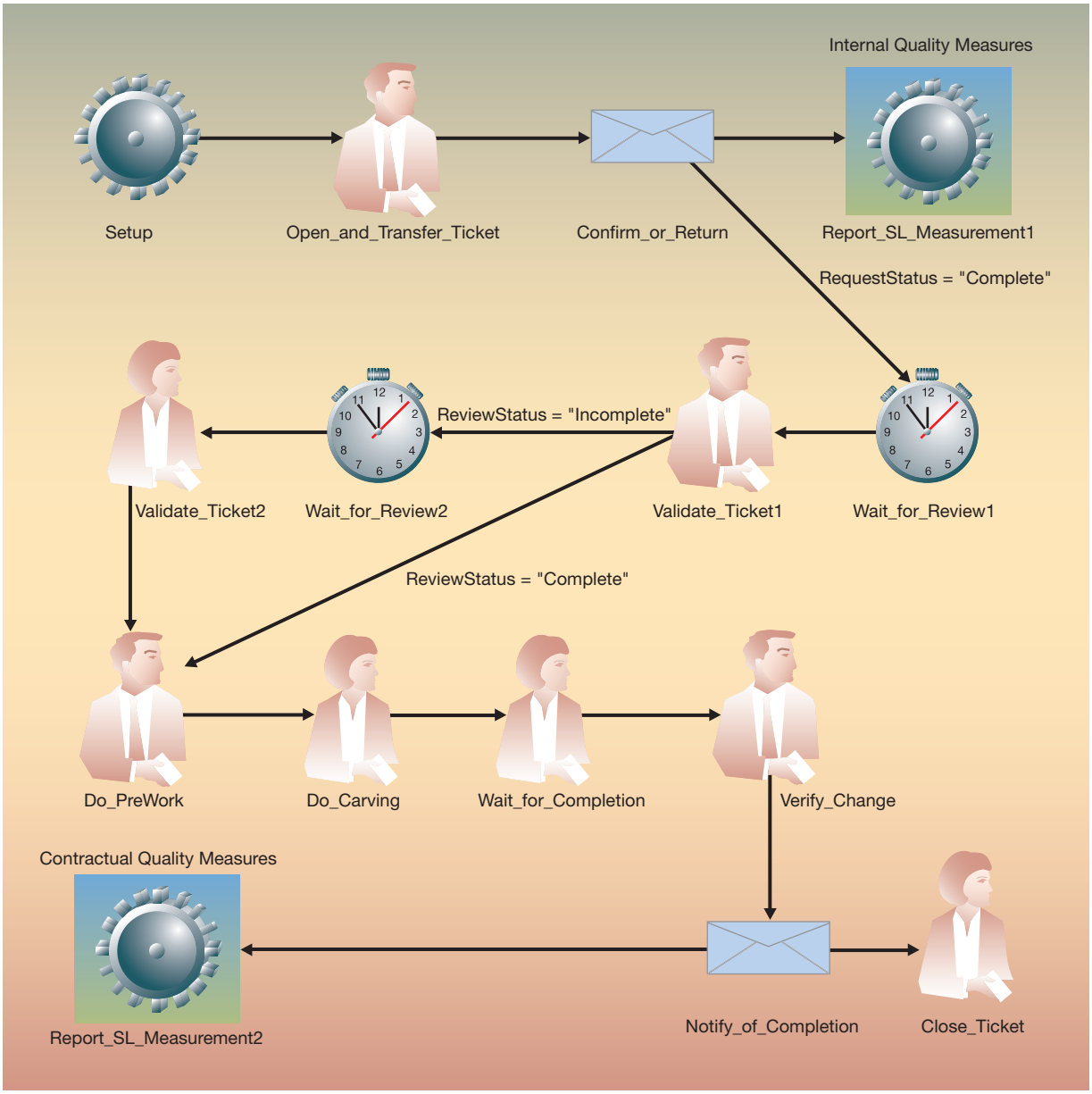


Figure 11 shows a sample on demand storage provisioning process captured as a workflow. The semiautomated process integrates several manual tasks (shown by person icons in the figure) with automated tasks performed by service management agents. The process reports measurement data for two service levels: a noncontractual service level for initial response time [Report_SL_Measurement1] and a contractual service level for overall response time to an on demand storage request [Report_SL_Measurement2].

Although the sample process is relatively simple, it incorporates a number of features commonly supported for directed task graphs in workflow engines:

- Looping (e.g., [Wait_for_Review1] is a loop of several tasks that are repeated until an exit condition is met)
- Conditional execution (e.g., [Wait_for_Review1] is only executed if the storage request is complete)
- Parallel execution (e.g., [Report_SL_Measurement2] and [Close_ticket] can be performed simultaneously).

The initial design of the scheduler focused on optimizing the use of human resources for process instances consisting of tasks in chain precedence order (i.e., a list of tasks performed in sequence with no loops, no conditional execution, and no parallel execution). This scheduling problem is already extraordinarily challenging under the chain precedence assumption, and we thus chose to implement this version initially. We recognize, however, that the chain precedence model of jobs and tasks is not always sufficient for modeling SLM processes, and generalizing this is one item on our list for future work.

We have created a method to dynamically convert a less restrictive task graph into chain precedence order at run time, based upon the SLM process management needs for the sample on demand resource-provisioning process. Although this conversion cannot necessarily be accomplished for an arbitrary task graph, it does allow us to support a larger and more realistic subset of task graphs than just those satisfying the chain precedence constraints.

In order to accomplish this run-time conversion to a chain precedence task list, the codified SLM process must be annotated at the workflow build time with information, such as the expected length of time to perform a task and the links that are expected or can be ignored for the purpose of resource allocation. Before a managed workflow process is deployed, a process analysis step is required to capture the process specification details needed at run time and to ensure that the process obeys the following restrictions:

- There can be no possible parallel execution of tasks that require scheduled resources. However, there can be parallel execution paths that contain only tasks which (1) do not require any scheduled resources and (2) have no immediate or subsequent links with tasks that do require scheduled resources. Such parallel paths can be defined by flagging the initial links as ignore for the purpose of resource scheduling. All subsequent links are automatically regarded as ignore links.
- A task can be the starting point for at most one unconditional link that is not flagged as ignore or one conditional link that is flagged as expected.
- A subprocess or block task must be flagged as transparent if any of its tasks require resource scheduling. The expected number of iterations for transparent blocks can be set to a number greater than one.
- Expected times must be supplied for all tasks that are not on an ignored path (that is, not the end point of an ignored link) with the exception of transparent-subprocess or block tasks for which expected times must be supplied for all contained tasks (subject to the same condition).

At run time, the Cross-SLA Process Execution Manager monitors the execution of each process instance that requires resource scheduling. It informs the scheduler of task completions and any changes to the expected task list for a process instance.

Conclusion

In the era of on demand computing, a leading utility computing provider must proactively: (1) maximize customer satisfaction with competitive service level reports (in terms of both coverage and attained quality), (2) minimize the exposed business impact of service level violations, and (3) lower the cost-to-quality ratio of executing SLM processes. In order to make timely adequate service management decisions based upon the exposed business impact of all action-demanding quality alerts, the provider's SLA management system must be capable of performing on demand intermediate service level evaluations with support for adjudication processes for contractual quality measures. The intermediate SLA compliance evaluation results must be as accurate and current as possible so that appropriate SLM processes can be executed in a timely manner.

SAM is a distributed SLA management system that can be seamlessly integrated into a utility-computing operating environment through SLM process automation and integration technologies. The system enables a utility computing provider to proactively fulfill its service-quality management commitments to all of its SLA contracts in accordance with business requirements. The SAM system would, among other benefits: (1) enable the deployment of an effective means of capturing and managing SLM-related contractual data as well as the provider's internal management data, (2) enable on demand intermediate evaluation of contractual service levels with

Table 1 Monthly refunds or premiums for missing or exceeding the SLA standard

Duration of SLA Downtime	Amount of Credit or Premium
More than 48 hours in a calendar month	100% credit of monthly charge
More than 36 hours in a calendar month	80% credit of monthly charge
More than 24 hours in a calendar month	60% credit of monthly charge
More than 16 hours in a calendar month	40% credit of monthly charge
More than 8 hours in a calendar month	20% credit of monthly charge
More than 4 hours in a calendar month	10% credit of monthly charge
4 hours or less in a calendar month	None
2 hours or less in a calendar month	10% premium of monthly charge
1 hour or less in a calendar month	20% premium of monthly charge
No downtime in a calendar month	30% premium of monthly charge

support for manual or automatic adjudication of quality measures, (3) assist service personnel and service management agents in ordering quality alerts based upon the exposed business impact over time, and (4) automate the prioritization and execution management of SLM processes, including assigning SLM tasks to service personnel using continual optimization technologies.

Preliminary event-driven, simulation-based evaluations of our SLM task and resource optimizing scheduler show SAM can significantly reduce the financial risk of service level violations relative to other common SLM job and workforce scheduling algorithms. Our development and internal pilot deployment experience with SAM suggests our business-oriented, process-centric approach to managing utility computing SLA contracts is both practical and useful.

Appendix: An Abridged Web Hosting SLA Contract

Provider's Service Level Agreement (SLA) standard for Customer's Web Hosting environment is less than four hours per calendar month of downtime, which is an availability of approximately 99.5 percent. This SLA objective applies to downtime caused by Provider in regard to operation of system software, loading of system software, hardware failure, backup and recovery of files, and connectivity from the server farm to the Internet and from the server farm to the Customer data center. This SLA objective specifically does not include failures caused by Customer, outages associated with contract maintenance provisions, failure of bandwidth connectivity, and external failures outside the Web-site Hosting Environment.

Availability for the purposes of the SLA objective is based either on Help Desk trouble-ticket information or Provider-detected downtime. For problems reported to the Provider Help Desk trouble-ticket system, opening of the Help Desk trouble ticket establishes the outage begin time; the problem resolved time, as documented in the Help Desk trouble ticket, establishes the end time for that particular outage. For Provider-detected downtime, the outage begin time is based on the first detection of any outage, and the end time for the outage is based on the problem resolve time. The amount of downtime in each calendar month will be totaled to determine any failure to meet the SLA objective.

Table 1 documents the monthly refunds or premiums associated with missing or exceeding the SLA standard for each calendar month. In no case shall more than the monthly charge be credited for downtime incurred in a single month.

Credits and premiums will be aggregated and settled on an annual basis. Premiums may be used to offset credits, but will not create an obligation on the part of a customer to pay more than those monthly charges documented in this statement of work (SOW) or subsequent transaction document pertaining to these services. Any payment due from Provider to Customer will be paid by January 31st of the following year.

The credit and premium schedule associated with this SLA and described herein did not commence until January 1, 1999; however, the Provider used best efforts to meet the SLA standard beforehand.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Object Management Group.

Cited references and note

1. L. Greenemeier, "Plugged In: Utility Computing Has Been Tested at a Few Companies. Will Others Follow the Pioneers?" *Information Week* (July 29, 2002).
2. R. Sturm, W. Morris, and M. Jander, *Foundations of Service Level Management*, SAMS Publishing, Indianapolis, IN (2000).
3. A. Hiles, *The Complete IT Guide to Service Level Agreements—Matching Service Quality to Business Needs*, Rothstein Associates Inc., Brookfield, CT (1999/2000).
4. N. Frey, R. Matlus, and W. Maurer, *A Guide to Successful SLA Development and Management*, Gartner Group Research, Strategic Analysis Report, Gartner, Inc., Stamford, CT (October 2000).
5. *A White Paper on Service Level Agreement,* ASPIC Best Practices Committee, Application Service Provider Industry Consortium (ASPIC), now a section of the Computing Technology Industry Association, Oak Terrace, IL (November 2000).
6. In order to illustrate the nature of SLA terms in this paper, representative clauses extracted from commercial SLA contracts are presented as italicized text within quotation marks.
7. D. Verma, *Supporting Service Level Agreements on IP Networks*, Macmillan Publishers, New York (1999).
8. *A Guide to the ASP Delivery Model,* ASPIC Best Practices Committee, Application Service Provider Industry Consortium (ASPIC), now a section of the Computing Technology Industry Association, Oak Terrace, IL (November 2000).
9. L. Lewis and P. Ray, "Service Level Management, Definition, Architecture, and Research Challenges," *Proceedings of the Global Telecommunications Conference (Globecom '99)*, Vol. 3 (1999), pp. 1974–1978.
10. R. Sturm, L. Erickson-Harris, and D. Onge, *SLM Solutions: A Buyer's Guide*, Enterprise Management Associates, Boulder, CO (July 2002).
11. *Ensuring Site Availability: Is Your Managed Service Provider Up to the Task?,* Harvard Computing Group Report, Harvard Computing Group, Westford, MA (2001).
12. P. Bhoj, S. Singhal, and S. Chutani, "SLA Management in Federated Environments," *Computer Networks* **35**, 5–24 (2001).
13. 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)* (September 2002), pp. 363–376.
14. M. Buco, R. Chang, T. Kosar, L. Luan, S. Shah, C. Ward, and P. Yu, "Managing eBusiness on Demand SLA Contracts in Business Terms Using the Cross-SLA Execution Manager *SAM," Proceedings of the 6th International Symposium on Autonomous Decentralized Systems (ISADS)* (April 2003), p. 157.
15. F. Griffel, M. Boger, H. Weinreich, W. Lamersdorf, and M. Merz, "Electronic Contracting with COSMOS—How to Establish, Negotiate and Execute Electronic Contracts on the Internet," *Proceedings of the 2nd International Conference on Enterprise Distributed Object Computing (EDOC)* (November 1998).
16. 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 28, 2003).
17. *ITAA's ASP SLA Resources,* Information Technology Association of America, Arlington, VA (2002).
18. *Unified Modeling Language (UML),* Object Management Group, Needham, MA.
19. *Service Level Agreement between JISC and UKERNA,* UKERNA (April 2001), http://www.jisc-tau.ac.uk/ukerna-sla.html.
20. T. Genovese and B. Jennings, *A Common Schema for the Internet White Pages Service,* Network Working Group, Internet Engineering Task Force, Request for Comment (RFC) 2218 (October 1997).
21. K. Bohrer and B. Holland, *Customer Profile Exchange (CPExchange) Specification,* (www.cpexchange.org) (October 2000), http://xml.coverpages.org.cpexchangev1_OF.pdf.
22. *The IBM Utility Management Infrastructure,* IBM Global Services, IBM Corporation (April 2003).
23. D. Kosiur, *Understanding Policy-Based Networking*, John Wiley & Sons, New York (2001).
24. D. Verma, *Policy-Based Networking: Architecture and Algorithms*, New Riders Publishing, Indianapolis, IN (2001).
25. X. Gu, K. Nahrstedt, R. Chang, and C. Ward, "QoS Assured Service Composition in Managed Service Overlay Networks," *Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003)* (May 2003), p. 194.
26. H. Smith and P. Fingar, *Business Process Management: The Third Wave*, Meghan-Kiffer Press, Tampa, FL (2003)
27. R. Peisl, *Delivering Business Agility and Service Level Management with Business Process Integration*, G325–1999, IBM Corporation (2000).
28. A. G. Ganek and T. A. Corbi, "*The Dawning of the Autonomic Computing Era*," *IBM Systems Journal* **42,** No. 1, 5–18 (2003).
29. J. Blazewicz, K. Ecker, G. Schmidt, and J. Weglarz, *Scheduling in Computer and Manufacturing Systems*, Springer-Verlag, Berlin (1993).
30. *Computer and Job-Shop Scheduling Theory*, E. Coffman, Editor, John Wiley & Sons, New York (1976).
31. M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Prentice-Hall, Englewood Cliffs, NJ (1995).
32. M. Garey and D. Johnson, *Computers and Intractability*, W. H. Freeman and Company, San Francisco (1979).
33. J. Wolf, M. Squillante, J. Turek, and P. Yu, "Scheduling Algorithms for Broadcast Delivery of Digital Products," *IEEE Transactions on Knowledge and Data Engineering* **13,** No. 5, 721–741 (2001).
34. J. Wolf, M. Buco, R. Chang, L. Luan, C. Ward, and P. Yu, *On Workforce Scheduling to Minimize SLA Contract Penalties,* Technical Report, IBM Corporation (2003).
35. R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK (1995).
36. T. Ibaraki and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, Cambridge, MA (1988).

Accepted for publication September 5, 2003.

Melissa J. Buco *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (mjbuco@us.ibm.com)*. Ms. Buco is a senior software engineer in the Services and Software department at the Watson Research Center. She received a B.A. in mathematics from Northeastern University and an M.S. in computer science from Columbia University. She received an IBM Outstanding Technical Achievement Award for her work on PCWARES. She has also worked in the areas of software engineering, emergency management, project management, and workflow.

Rong N. Chang *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (rong@us.ibm.com)*. Dr. Chang received his Ph.D. in computer

science and engineering from the University of Michigan at Ann Arbor in 1990. Before joining IBM in 1993, Dr. Chang was with Bell Communications Research creating advanced personal ubiquitous application services for ATM-based broadband networks. He is now manager of Network Hosted Application Services at the Watson Research Center. He has received several awards at IBM, including an Outstanding Technical Achievement Award and a Technical Excellence Award. At IBM he is a leading contributor to: (1) the SLA management component in the IBM Universal Management Infrastructure, (2) real-time credit card processing and software download service at ibm.com, (3) the common registration infrastructure for IBM's Internet Web sites, and (4) the first hyperlink-aware search engine (named WebCat). His research interests include quality-assured on demand services, distributed computing systems, workflow-centric computing, hypermedia information management, and personal pervasive computing and communication.

Laura Z. Luan *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (luan@us.ibm.com).* Dr. Luan received her Ph.D. in electrical engineering from Keio University in Japan. She was a scientific staff member at Nortel Research Division from 1990-1996 working on SONET (Synchronous Optical Network) network synchronization and digital transmission systems. She joined IBM in late 1996 and worked on the High-Definition Television (HDTV) Broadcast Technology Collaboration project in the area of distributed command and control for real-time high-bandwidth multimedia streaming over ATM (asynchronous transfer mode) networks. Her current research interests are in the areas of utility computing infrastructure management for on demand services, in particular, contract execution management and resource management.

Christopher Ward *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (cw1@us.ibm.com).* Dr. Ward received his Ph.D. in computer science from the University of Florida in 1988. He joined IBM in 2000 and is most recently responsible for an extensible SLM data management framework to represent the complex relationships required for proactive SLA management. Since joining IBM he has received various achievement awards, has chaired selected standards committees, and has published several technical papers. Prior to joining IBM, Dr. Ward was Head of the Advanced Networking and Distributed Control Group and a Senior Member of the Technical Staff at Sarnoff Corporation. During this period he was responsible for development of distributed control and compressed bit stream splicing technologies in support of professional broadcast studios. Prior to his employment in corporate research facilities, he held academic appointments at City University of New York and Auburn University. Dr. Ward has published over 40 papers addressing a variety of computer science problems, is the author or co-author of numerous patents, and is a Senior Member of the IEEE.

Joel L. Wolf *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (jlwolf@us.ibm.com).* Dr. Wolf received his Ph.D. degree from Brown University in 1973 and his Sc.B. degree from the Massachusetts Institute of Technology in 1968, both in mathematics. He is currently a research staff member at the Watson Research Center, with interests in mathematical optimization. He has received two IBM Outstanding Innovation Awards, is a Master Inventor at IBM, and is a Fellow of the IEEE. He has also been an Assistant Professor of Mathematics at Harvard University, as well as a Distinguished Member of the Technical Staff and manager at Bell Laboratories.

Philip S. Yu *IBM Research Division, Thomas J. Watson Research Center, 19 Skyline Drive, Hawthorne, New York 10532 (psyu@us.ibm.com).* Dr. Yu received a B.S. in electrical engineering from National Taiwan University, an M.S. and Ph.D. in electrical engineering from Stanford University, and an M.B.A. degree from New York University. He is currently manager of the Software Tools and Techniques group at the Watson Research Center. His research interests include data mining, Internet applications and technologies, database systems, and multimedia systems. He has published more than 350 papers in refereed journals and conferences. He holds or has applied for 234 U.S. patents. Dr. Yu is a Fellow of the ACM and the IEEE. He is the Editor-in-Chief of *IEEE Transactions on Knowledge and Data Engineering.* He is also an associate editor of *ACM Transactions on Internet Technology* and of *Knowledge and Information Systems.* He has received several IBM and external honors, including Best Paper Award, two IBM Outstanding Innovation Awards, an IBM Outstanding Technical Achievement Award, two IBM Research Division Awards, and the 73rd plateau of Invention Achievement Awards. Dr. Yu is an IBM Master Inventor.