RC24317 (C0707-009) July 20, 2007
Computer Science

IBM Research Report

Service Obligations Management in IT Outsourcing

Liang Liu[1], Hui Lei[2], Dongjun Lan[1], Raymond E. Rose[2], J. S. Seymour[3], Murthy V. Devarakonda[2]

[1]IBM Research Division
China Research Laboratory
Building 19, Zhouguancun Software Park
8 Dongbeiwang West Road, Haidian District
Beijing, 100094
P.R.C.

[2]IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

[3]IBM Global Business Services
55 Pyrmont Street
Pyrmont NSW 2009
Australia

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

Service Obligations Management in IT Outsourcing

Liang Liu[1], Hui Lei[2], Dongjun Lan[1], Raymond E. Rose[2], J.S. Seymour[3], Murthy V. Devarakonda[2]
IBM China Research Laboratory[1]
IBM T.J. Watson Research Center[2]
IBM Global Business Services[3]
{liuliang, landj}@cn.ibm.com[1], {hlei, rer, mdev}@us.ibm.com[2], {jseymour}@au1.ibm.com[3]

***Abstract*-In outsourcing, service obligations broadly refer to the duties and responsibilities of the service provider related to the provision of services. It is critical that the provider clearly understands and properly implements its complete service obligations to the client. The BlueMoon project in IBM develops a novel approach to service obligations management in the context of IT outsourcing. BlueMoon conceptualizes different kinds of service obligations through a formal obligations model. It provides an overarching obligations configurator tool for gathering and validating obligation requirements throughout the outsourcing lifecycle. It further facilitates the implementation of service obligations by automatically binding them to the IT architecture planning and provisioning process and to their uses in various service documents. Work is under way to monitor service delivery operations and verify obligations compliance. Addressing a key challenge in IT outsourcing, BlueMoon promises to significantly improve the client's satisfaction, the health of the outsourcing account, and the provider's bottom line.**

I. INTRODUCTION

Outsourcing is the practice where a provider takes responsibility for delivering part of a client's operations and manages it on the client's behalf, normally to an agreed cost and level of service. It builds on the premise that the external provider, being an expert in the field, could perform the same task effectively for many organizations and achieve economies of scale. Early outsourcing engagements were confined to low-impact functions such as cleaning and catering, and turned out to be very viable and successful. In today's highly dynamic and competitive business environment, there is a strong move towards outsourcing operations like IT and logistics that are more business-critical or give competitive advantages in certain ways.

IT outsourcing involves the management of a company's computer applications and IT systems, with a possible transfer of the IT employees and IT assets from the outsourced company to the provider. It promises to reduce IT costs and improve operational performance, provides the opportunity for companies to transition and transform to more advanced IT infrastructure and applications, allows them to focus on their core business competencies and gain access to deep technology expertise.

A typical IT outsourcing arrangement spans multiple years. Its lifecycle comprises some distinct stages: engagement, transition, transformation, and delivery. During the engagement stage, the client's requirements are gathered; the service scope is defined, along with the IT infrastructure and tools that will be used to deliver the services. The transition stage starts once the client has indicated the intent to award the contract; the service provider gears up to take over client operations as-is and prepares to deliver contracted services. The service start date marks the beginning of the transformation stage, when the client's IT environment is converted to the target architecture defined during engagement and new service tools are deployed as necessary. The delivery stage is for steady-state operations; the provider runs the client's IT operations in accordance with the contract and commits to service levels.

In outsourcing, the term *service obligations* broadly refers to the duties and responsibilities of the provider related to the provision of services. It is critical that the provider clearly understands and properly implements its service obligations to the client. However, that is easier said than done. While some service obligations may be explicitly spelled out in legal documents like contracts based on discussion in the engagement phase, more detail-oriented obligations have to be discovered during transition and transformation stages by talking to subject matter experts or going through the client's legacy materials. New obligations may be requested and established in the course of steady-state delivery. Still other obligations are so subtle and implicit that they may not be recognized until serious consequences have resulted from failure to implement them. Current practices don't use a common repository to maintain known obligations. Obligations information is scattered in a variety of places like contracts, project control books, process interface manuals, and desk procedures. Since such documents are intended for human users (e.g., project managers, service personnel), obligations information is represented as free-form text and is not machine-interpretable.

Existing practices have serious limitations. Significant human effort is required to find, enter, and implement service obligations. The same information may have to be input multiple times by different role players at different points of the outsourcing lifecycle, resulting in duplicate work and redundant data. Text representation of obligations can lead to ambiguity. Information captured in different documents may not be consistent. Some obligations may not be recorded or get lost in the process. Failures to implement obligations may go

unnoticed until a disaster strikes. The net outcome is client dissatisfaction, frequent contract re-negotiation, account erosion, and costly services.

The *BlueMoon* project in IBM is developing a novel approach to service obligations management. It set out to answer the following questions:

- **How to impose structure on the unstructured?** There is a large volume of obligations information in any outsourcing arrangement. How to model and represent the obligations and their interrelationships so that they can be better managed and reasoned about?
- **How to maintain information continuity?** During the outsourcing lifecycle, one obligation requirement may manifest in different forms of implementation. How to facilitate the mapping from obligation requirements to their implementation? How to handle the semantic gap between the requirements and implementation?
- **How to verify compliance?** Obligations may be performed either manually or programmatically. How to ensure that all obligations have been complied with? How to interpret the financial consequences of anomalies and facilitate corrective actions?

Figure 1 gives an overall architecture of BlueMoon. BlueMoon consists of three functional components. The first component is *Obligations Capturing*. BlueMoon conceptualizes different kinds of service obligations through a formal obligations model. It provides an obligations configurator tool to gather obligation values and validate their consistency and integrity. It further maintains in a central repository all obligation values in an outsourcing account. The second functional component is *Obligations Implementation*. This component is in turn made of two sub-components. The *Service Planning and Provisioning* sub-component deals with the planning and provisioning of the underlying IT architecture and the service tools in accordance with obligations requirements. The *Service Document Generation* sub-component maintains linkage between obligations and their references in various service documents. The service documents include both human-facing documents such as contracts and desk procedures and computer-oriented documents such as tool configuration scripts. The goal of this sub-component is to programmatically populate and update relevant parts of the service documents based on data in the obligations repository. The third functional component in BlueMoon is *obligations monitoring*. BlueMoon intends to monitor the service delivery operations to verify the compliance of the obligations and provide visibility and alerts through a dashboard.

This paper provides a status report of the BlueMoon effort. Section 2 presents the service obligations model and the obligations capturing mechanism. Sections 3 and 4 are devoted to the two aspects of obligations implementation respectively, i.e., service planning and provisioning and service document generation. Obligations monitoring is work in progress and will be discussed in a future paper. Section 5 briefly discusses related work. Section 6 concludes the paper.

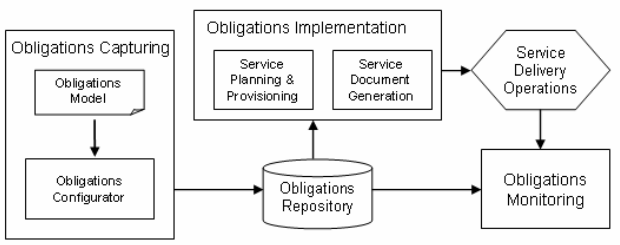

Figure 1. Overall System Architecture

II. Obligations Modeling and Capturing

BlueMoon models service obligations with respect to a service catalog. Service catalogs are commonly used by IT service providers to standardize the service offerings and comprise a classification of services. They are typically organized into a hierarchy based on skills competencies: the higher levels of the hierarchy correspond to broad competencies and the lower levels specific skills within a competency. One service catalog, for instance, may define a hierarchy of three levels – service lines, service segments, and service elements. An example of a service line is "storage management". An example of a service segment within this service line is "media management" and an example of a service element within this service segment is "backup and restore". Each node in the service hierarchy is said to define a service, regardless of the level it belongs to.

BlueMoon conceptualizes four kinds of service obligations.

1) *Service functions:* finer-grained service activities that support the lowest level of services in a service catalog. One service function for the "backup and restore" service element above can be "creating backup accounts for users".
2) *Workload characterization:* the workload that must be accommodated by the service provider. For example, there can be up to *500* concurrent requests for file restore.
3) *Operational parameters:* input parameters to service tool configuration and delivery processes. For example, backup images should be retained for *2* years.
4) *Service levels:* quantifiable expectation of service quality. For example, each backup image on disk must be restored within *1* minute, and each backup image on tape must be retrieved within *10* minutes.

Figure 2 shows a metamodel that relates the concepts discussed so far. A service catalog consists of a collection of services, which form a hierarchy through the parent-child relationship. Each service is optionally associated with a set of service functions. Both services and service functions may be coupled with quantifiers that represent workload parameters, operational parameters, or service level agreements (SLAs). Workload parameters and operational parameters are sub-types of service parameters and are constrained to a predefined data type. The data types can be basic types like string, integer, and

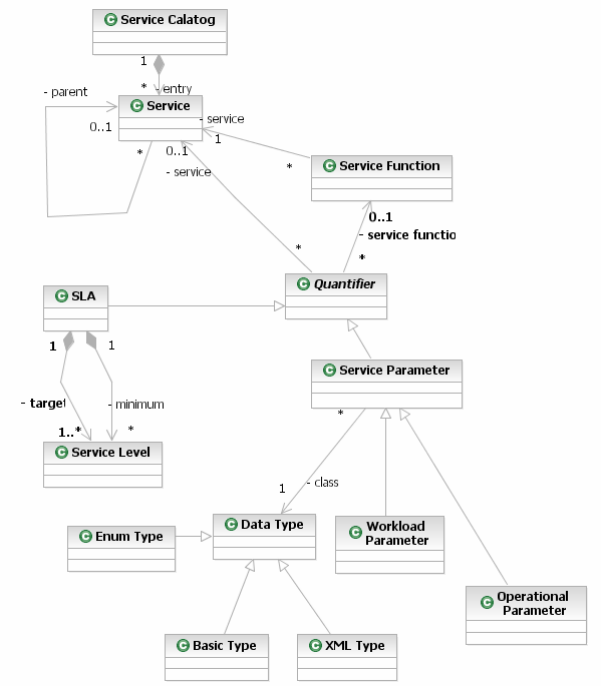

Figure 2. Service Obligations Metamodel

boolean. They can also be enumeration types or more sophisticated ones defined through XML. An SLA consists of a collection of target service levels and a collection of minimum service levels. Failure to meet a minimum service level will result in financial penalty to the service provider. While it makes sense to define service functions for the lowest-level of services only, quantifiers may be defined for service functions, or any levels of services.

A provider's standardized service capabilities can be specified by a *service reference model*, which is an instantiation of the above metamodel. A service reference model defines the specific services offered by the provider, the detailed service functions that may be selected by a customer, and predefined parameters related to workload, operations, and service levels which a customer can configure. It is relatively static in nature and evolves as the provider offerings evolve.

A *service solution* is an instantiation of a service reference model. It is bound to a specific outsourcing arrangement and captures the data that describes particular service obligations that the service provider has with an individual client under a service contract. As such, it is relatively dynamic in nature. Service solutions vary by service contracts and can evolve throughout the service lifecycle.

A service solution includes a selection of services from the service catalog, a selection of service functions for selected services, and the values for the service parameters defined for those services and service functions. Within a solution, one or more service function instances may be defined for a selected service function and each service function instance may carry its own unique service parameters and service levels. This allows different obligations with regard to different subsets of a client's assets to be modeled. For example: the "perform file backup" service function might have two instances: one for system files and one for application files. The two instances can have different backup schedules as operational parameters.

Both a relational database representation and an XML-based representation have been implemented for obligation models and instances. In the relational database representation, the service reference model and the service solutions are respectively stored across a collection of relational tables. The tables that contain the service reference model are relatively static and serve the same content to all outsourcing accounts. The tables for the service solutions contain solutions for all accounts and are properly indexed by account ID. Service personnel for an account are granted access to the solution for that particularly account only. The advantages of the database representation are that it is easy to implement and is efficient to read and write to. On the other hand, the relational data model is intended for homogeneous and well structured data. It is not always natural for maintaining highly diversified data. For example, the current implementation has to store all types of service parameter values as strings in the database and performs type conversion out of band.

In the XML representation, the obligations metamodel in Figure 2 is defined as an XML schema with abstract types for the four kinds of obligations. The service reference model is also defined as an XML schema, where specific obligations are defined as concrete extensions of the abstract types in the metamodel schema. A service solution is represented as an XML document conforming to the service reference model schema. The flexibility of XML is well suited for the tasks, as different outsourcing services may be associated with very diverse sets of obligations and different clients may also have very different requirements on service scope and obligations. The flexibility, on the other hand, comes at a small cost of simplicity and efficiency.

BlueMoon provides an obligations configurator tool that allows authorized users to create, view and edit a service solution. The user interface of the tool consists of two tabs: a services tab and an obligations tab. From the services tab, the user is presented with a view of the entire service catalog augmented with service functions, organized in the form of a tree. To select a service or service function to the solution, the user highlights the respective item and indicates the "Add to Solution" action. From the obligations tab, the user is able to edit the service parameters and service levels associated with a selected service or service function. In the case of a service function, the user is also able to create one or more service function instances to the solution and to edit the obligation values associated with each instance. The user may browse all the obligations in a solution by navigating through the service hierarchy. Alternatively, he or she can view the obligations by their kinds. A screen shot of the tool is given in Figure 3.

The obligations configurator is a Java application developed on top of the Eclipse rich client platform. Hibernate is used to implement the object-relational mapping layer and Eclipse EMF is used to provide Java access to XML objects.

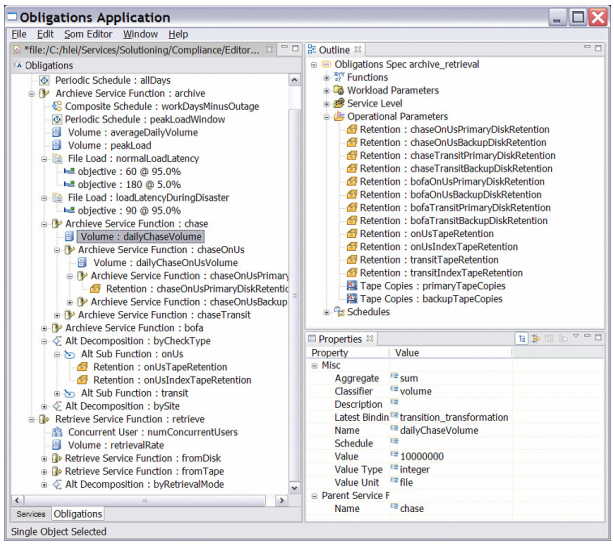

Figure 3. Obligations Configurator User Interface

III. Service Planning and Provisioning

One aspect of the implementation of service obligations is the planning and provisioning of an appropriate IT infrastructure in a data center for service delivery to the client. The IT infrastructure typically will host a variety of services, each of which has its own set of obligation requirements. Planning and provisioning of the IT infrastructure therefore must take into account all such requirements in a holistic manner. BlueMoon uses the following procedure. First, as the basis of planning, a model of the data center is created. In the model, components of data center, e.g., servers and network devices, are organized into different resource pools (the server pool, the network device pool etc). Second, a deployment planner is used to generate an infrastructure deployment plan from the service obligations and the data center model. Third, using an autonomic provisioning tool, the infrastructure deployment model is provisioned from the resource pools, and deployed in the actual data center.

The data center model is an abstract representation of the data center. It identifies the components of the data center and captures their relationships. An example data center model is given in Figure 4. The data center objects can be network-related objects, servers, system software, or applications. Network-related objects include switches, routers, VLANs, subnets and the switch fabric. Servers refer to various machines that can perform computation, for instance, blade servers, main frames. Servers can be physical servers or virtual servers. Virtual servers typically share resources with other systems. A cluster is a collection of servers to enable high availability and reliability for applications that are important to business. Resource pools within the data center model are containers or reservoirs of available servers. When more capacity is required, a server can be provisioned from the appropriate resource pool. When the demand is low, a server can be returned to the resource pool, so that it is available to other applications. Applications are owned by customers and are deployed in data center. Software products include different software that will be installed in data center. Software stack is a bundle of software images that will be installed for a specific customer application. Boot server is used for the automatic cloning, deployment and recovery processes on a target computer. The data center model further includes key attributes of components, such as performance estimation of servers for capacity sizing.

The mapping from service obligations to an infrastructure deployment plan is handled by a deployment planner. The planner first extracts obligations values, and gathers additional information such as hardware and software characteristics. The planner then obtains the resource availability information from the resource pool and translates obligation requirements to resources required, considering resource performance data and service demands at different levels. Resource performance data is collected either from online data repositories or from system administrators. The service demands are used to predict overall system throughput and response time under certain resource utilization, based on a queuing network model. The deployment planner performs its job through a carefully orchestrated workflow. Using the workflow, various capacity-sizing tools are used for deriving the resources needed, for example, Opera for open system, and zPCR for main frames. Finally the planner generates a deployment recommendation. Deployment planning is a complicated but well studied subject. Structured representation of service obligations reduces the demand for human effort in this process by making substantial amount of required input readily available to the planner.

The deployment plan is carried out in the data center by provisioning and deploying required servers, network, software, and applications. Many data centers employ automatic provisioning capabilities, for example, a combination of the Tivoli Intelligent Orchestrator and the Tivoli Provisioning Manager. The provisioning process, choreographed via the automatic provisioning tools, processes the deployment plan, and provisions infrastructure components one at a time.

IV. Service Document Generation

The Obligations Binding component in BlueMoon takes obligations information from a service solution and automatically generates parts or the entirety of files needed for service operations. Such files fall into two categories: documents intended for human users and configuration scripts to be used by computers for configuring the underlying IT infrastructure and the services tools. BlueMoon uses different approaches to generating these two kinds of files.

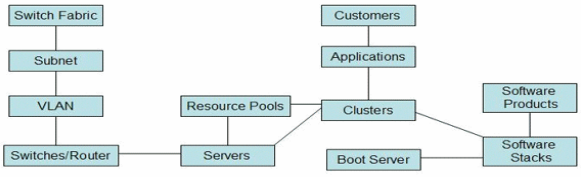

Figure 4. Example Data Center Model

Human-facing documents are generated from boilerplate document templates with client-specific content drawn from, or derived from, the obligations information. This task involves more than simple insertion of obligation values into the output document. As each account is at liberty to dictate the format of the documents used, it is necessary to permit some degree of customization in the format. Depending upon the absence, presence, or content of certain obligations, sections of the document may need to be formatted in entirely different ways.

In order to minimize disruption of existing service practices, the document templates are patterned after the layout and style of existing manually created documents. Virtually all of these documents were written with Microsoft Word. It became clear that the task could be accomplished most effectively by using a template with embedded VBA code. This language was designed to be integrated with Office products and its technology was mature.

Figure 5(a) shows an excerpt of a document template. The template is for the part of a formal contract called Schedule A, which details the services to be provided. The template assumes service obligations are represented in an XML document and references them via XPath expressions. It mixes control directives within brackets and rich text to be preserved in the output document. This particular example finds all obligation values that describe the workload of retrieval volume, and outputs them in different formats based on whether there is a text description associated with the value. A sample output is given in Figure 5(b). Since the templates are themselves Microsoft Word documents, rich text formatting ranging from lists and tables to fonts and colors can be easily specified and is retained in the output.

Two design alternatives were considered, but eventually rejected. XSLT style sheets are oriented towards XML transformation and are not suitable for formatting rich-text Word documents. On the other hand, Microsoft Word provides some XML capabilities in so called Smart Document support. Unfortunately it does not provide all the functions and flexibilities required to bind obligation values to Word document.

Automatic generation of the second category of documents - configuration scripts – differs from the automatic generation of human-facing documents in two ways. First, the configuration scripts are intended to be consumed by computer systems. There is no sophisticated styling or formatting as in Microsoft Word documents. This makes it possible to adopt a simpler approach to creating configuration scripts than using VBA. Second, the syntax and semantics of the configuration scripts are determined by the corresponding service tools. There is a large number of service tools out there. Even for the same service task, there are usually multiple vendor products available. These preclude developing a single configuration generator that covers all service tools. Instead, a plug-in framework is provided that allows tool-specific configuration generators to register themselves and be invoked at appropriate points of the service lifecycle and when a change of relevant obligations values is detected.

The current BlueMoon prototype includes a plug-in for generating the configuration scripts of the Tivoli Storage Manager (TSM), which is a client-server software for file backup and archival. The plug-in extracts from an XML-based service solution document parameter values related to backup schedule, data retention periods, media types, and rotation to offsite storage etc. It then creates the TSM server and client option files accordingly. In the case when different options are specified for different file groups, the plug-in creates multiple policy groups and associates them with the appropriate files.

One challenge a plug-in like the one for TSM must address is semantic impedance. Obligation values captured by the obligations configurator may be at a different level of abstraction from what is required to configure the service tools. For instance, the captured obligations may require that applications files be backed up on a daily basis and the system files on a weekly basis. However, the TSM requires explicit identification of file names or name patterns. Bridging this form of semantic gap is left to individual plug-ins. In the case of the TSM plug-in, a discovery tool Galapagos [13] is used to associate individual files to applications or infrastructure components in order to classify them as application files or system files.

V. Related Work

To the best of the authors' knowledge, the work presented in this paper is the first attempt at systematically managing the broad notion of service obligations. However, an important subset of service obligations, i.e., service-level agreements (SLAs), has been extensively studied. This section briefly discusses some representative work in SLA management.

Buco et al presents an SLA execution manager that enables a service provider to effectively execute SLA contracts [1]. The system helps the service personnel effectively capture contractual or non-contractual service level data, automates the

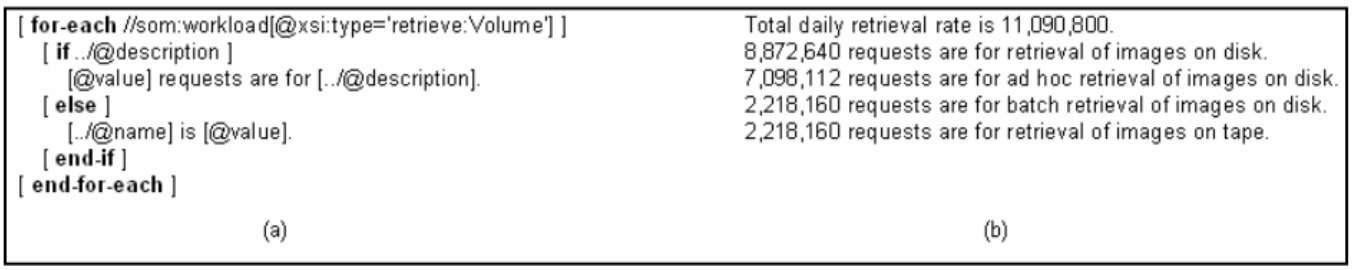

Figure 5. Example Document Template and Instance

assignment of SLA management tasks to service personnel, and assists service personnel in prioritizing the processing of such tasks per service level objectives. Buco's system and BlueMoon both support the capturing and the structured representation of SLAs. While the first system focuses on the SLA execution during service delivery phase, BlueMoon addresses the implementation of broader service obligations during service transition and transformation phases.

Garg etc. describe an SLA framework for QoS provisioning and capacity allocation, based on a three-tier pricing model [2]. With such a framework, users are no longer bound to fixed price and fixed capacity. They are allowed to dynamically relinquish parts of their capacities or buy additional capacities based on their needs.

The work by Lee et al [3] is concerned with the SLA monitoring of network access services. It proposes a mapping from service level parameters to network performance metrics, and discusses a general system architecture for monitoring the quality of services offered by network, Internet and application service providers.

The design goal of the Fresco system is to facilitate configuring extensible SLA management systems using Web Services [4]. Fresco uses an XML-based specification language to support contract-specific SLA terms as well as extensions of the deployed SLA management software. It further uses a template-based approach to communicate with other Web Services to accommodate various input and output formats.

Generally, SLA specifies the level of service that is expected during its term and is used by vendors and customers as well as internally by IT shops and their end users. SLAs can specify bandwidth availability, response times for routine and ad hoc queries, response time for problem resolution (network down, machine failure, etc.) as well as attitudes and consideration of the technical staff. However, *service obligations* broadly refers to the duties and responsibilities of the provider related to the provision of services, not only limited to the scope of SLA as discussed. It can be thought SLAs are important components of *service obligations*.

VI. Conclusion

Complete capturing and implementation of service obligations are critical to the success of any outsourcing arrangement. They have a direct impact on the client's satisfaction, the health of the outsourcing account, and the provider's bottom line. While prior work has been focusing on the structured management and automated execution of SLAs, BlueMoon conceptualizes the much broader notion of service obligations and advocates a systematic approach to the management of service obligations. BlueMoon identifies four different kinds of service obligations and unifies them in a formal data model. It provides an obligations configurator tool for gathering obligation requirements throughout the outsourcing lifecycle, and maintains them in a single trusted data repository. It further facilitates the implementation of service obligations by automatically binding them to the IT architecture planning and provisioning process and to their uses in various service documents. On-going work in BlueMoon includes the development of the obligations monitoring component and further validation of the system via supporting additional obligations types.

Addressing a key challenge in IT outsourcing, BlueMoon promises to substantially improve the quality and efficiency of outsourced IT services.

References

[1] Melissa J. Buco, Rong N. Chang, Laura Zaihua Luan, Christopher Ward, Joel L. Wolf, Philip S. Yu, Tevfik Kosar and Syed Umair Shah. Managing eBusiness on Demand SLA Contracts in Business Terms Using the Cross-SLA Execution Manager SAM. ISADS 2003 - International Symposium on Autonomous Decentralized Systems. IEEE, November 2002.

[2] Garg, R. Saran, H. Randhawa, R.S. Singh, M. IBM India Res. Lab., New Delhi, India: A SLA Framework for QOS Provisioning and Dynamic Capacity Allocation, In Proceedings of Tenth International Workshop on Quality of Service, May 2002.

[3] Hyo-Jin Lee, Myung-Sup Kim, James W. Hong, Gil-Haeung Lee. QoS Parameters to Network Performance Metrics Mapping for SLA Monitoring. APNOMS 2002.

[4] Christopher Ward, Melissa J. Buco, Rong N. Chang, Laura Zaihua Luan, Edward C. So and Chunqiang Tang. Fresco: A Web Services based Framework for Configuring Extensible SLA Management Systems. ICWS'05 - IEEE International Conference on Web Services. IEEE, April 2005.

[5] http://www-935.ibm.com/services/us/index.wss/itservice/so/a1000414

[6] Hiles, A., "The Complete IT Guide to Service Level Agreements - Matching Service Quality to Business Needs," *Rothstein Associates Inc.*, Brookfield, Connecticut, USA, 1999/2000 Ed.

[7] Frey, N., Matlus, R. and Maurer, W., "A Guide to Successful SLA Development and Management," Gartner Group Research, Strategic Analysis Report, October 16, 2000.

[8] H. Foster, "Mapping BPEL4WS to FSP, Technical Report," Imperial College, London 2003.

[9] M. Hafner and R. Breu, "From Inter-Organizational Workfows to Process Execution: Generating BPEL from WS-CDL," presented at ACM / IEEE 8th International Conference on Model Driven Engineering Languages and Systems, Montego Bay, Jamaica, 2005.

[10] E. Lupu and M. Sloman, "Conflict Analysis for Management Policies," presented at Proceedings of the Fifth IFIP/IEEE International Symposium on Integrated Network Management, San Diego, CA, 1997.

[11] M. V. Sinderen, L. F. Pires, C. A. Vissers and J. Katoen. A Design Model for Open Distributed Processing Systems. In Computer Networks and ISDN Systems, North-Holland pub., pp.1263-1285, 1995.

[12] http://www.service-level-agreement.net/

[13] K. Magoutis, M Devarakonda, and K. Muniswamy-Reddy, "Galapagos: Automatically Discovering Application-Data Relationships in Networked Systems" (short paper), in Proceedings of 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, Germany, May 21-25, 2007.

[14] Haitao Xia, Gehan Wu, Fulfilling SLA measurability via two-phased QoS metrics integration, Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE.

[15] http://www.covansys.com/what/amdo.htm