SLAng: A Language for Defining Service Level Agreements*

D.Davide Lamanna, James Skene and Wolfgang Emmerich
Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
{D.Lamanna|J.Skene|W.Emmerich}@cs.ucl.ac.uk

Abstract

Application or web services are increasingly being used across organisational boundaries. Moreover, new services are being introduced at the network and storage level. Languages to specify interfaces for such services have been researched and transferred into industrial practice. We investigate end-to-end quality of service (QoS) and highlight that QoS provision has multiple facets and requires complex agreements between network services, storage services and middleware services. We introduce SLAng, a language for defining Service Level Agreements (SLAs) that accommodates these needs. We illustrate how SLAng is used to specify QoS in a case study that uses a web services specification to support the processing of images across multiple domains and we evaluate our language based on it.

1 Introduction

The term 'electronic business', or 'e-business', refers to the execution of transactions of commercial significance in a distributed computing context, involving an organisation, its clients and its partners.

E-business is impeded by technical integration barriers. Recently, standardisation processes have begun to catch up with commercial development and functional integration is being enabled by two forces: Standardisation of component-based middleware architectures, and standardisation of communication protocols, particularly those based on Internet communication protocols and data formats. These technologies are complementary, and when employed in a client/server situation their use is often termed Application Service Provision (ASP) or web-service provision, if HTTP is used as the underlying transport protocol.

Functional integration may also include the provisioning of infrastructure by one organisation for another, as in the case of Internet Service Provisioning (ISP), Storage Service Provisioning (SSP) and application hosting. This provisioning relies on the use of standardized and established architectures and technologies.

Unfortunately, combining functionality is not the only requirement for e-business integration. Non-functional, quality requirements must also be met. Moreover, businesses must initially meet, negotiate the terms of their collaboration, have some confidence that the services that they purchase will meet their requirements, and that they in turn can meet their client's expectations. Efforts have been made to establish business-to-business marketplaces, in which application services can be traded, and we review these in Section 2. Our work stands in the context of these efforts, but addresses the need for description and negotiation of QoS properties.

The novel contribution of this paper is a reference model for inter-organisational service provision at storage, network, middleware and application level (Section 3). The model provides the basis for the definition of SLAng language, presented in Section 4. SLAs capture the mutual responsibilities of the provider of a service and its client with respect to non-functional properties. SLAng meets multiple objectives: It provides a format for the negotiation of QoS properties; the means to capture these properties unambiguously for inclusion in contractual agreements; and a language appropriate as input for automated reasoning systems or QoS-aware adaptive middleware. We have tried out the expressiveness of SLAng using a case study (CPXe) that supports the implementation of web services for image processing (Section 5). Section 6 critically evaluates SLAng and in Section 7, we summarise our contributions and discuss future directions for our research.

*This work is partly funded through the EU IST Project 34069 (TAPAS) and Kodak.

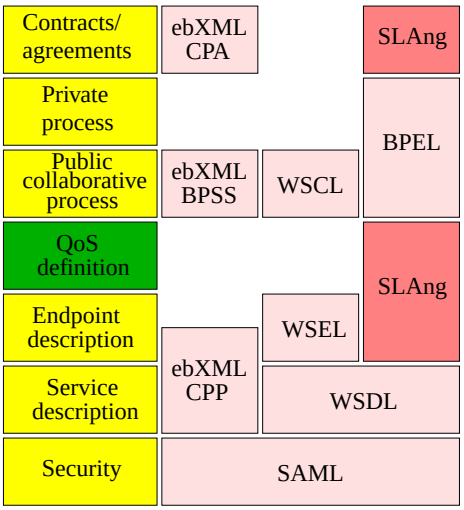

Figure 1. e-Business automation standards

2 Related work

A large number of industry standards have emerged that support the construction of distributed systems using web services and distributed component technologies, such as the Java 2 Enterprise Edition or the CORBA Component Model. These include WSDL and SOAP [10] for defining interfaces to web services, BPEL to define business processes and ebXML to define electronic business transactions. Figure 1 shows an overview of these standards [1] and how our language SLAng for service level agreements complements them. SLAng goes beyond them as it not only provides descriptions of quality at the application level, but also contractual agreements that are necessary when different ISO/OSI layers of a deployment are spread across multiple organizations.

The ISO/ODP trading function [2] and its various incarnations, for example the CORBA Trading Service [9] provide for quality of service definition. However, such traders define QoS at a single level of abstraction. Conversely, we allow for appropriate QoS definitions at different levels of abstraction, including the network level, the middleware level and the application level.

Significant research about QoS management and QoS-aware networks has been carried out by the TEQUILA project [8]. TEQUILA specifies and implements a set of service definition and traffic engineering tools to obtain quantitative end-to-end QoS guarantees at the network layer through careful planning, dimensioning and dynamic control of scalable and simple qualitative traffic management techniques within the Internet, such as differentiated services [4]. Their Service Level Specification proposal submitted to the Internet Engineering Task Force (IETF) is one of our starting points since we assume an SLS expressed with the parameters proposed in [8] for SLAs regarding networking services. The IETF are developing protocols and mechanisms for negotiating, monitoring and enforcing SLSs, and to ensure that the network can cope with the contracted SLSs. The efforts of the IETF, though, are based at the socket level (which only includes communication resources) rather than at the distributed object level (which includes communications, processing, and storage resources). As such, they cannot address end-to-end QoS issues at higher levels of abstraction.

For QoS-aware middleware, related work has been done by the Quality Objects (QuO) group ([7], [6]). QuO is a framework for providing QoS in network-centric distributed applications, ranging from embedded applications to wide area network applications. QuO bridges the gap between the socket-level QoS being specified, researched, and provided by a number of organizations and the distributed object level commonly used to write distributed systems. QuO has the merit of having raised the level of abstraction for QoS specification, but it is still not sufficient to allow provision QoS services given the diversity of distributed heterogeneous environments.

HQML (Hierarchical QoS Markup Language) [5], an XML-based specification language, addresses this issue. HQML enhances distributed applications on the World Wide Web with QoS capability. It allows the specification of all kinds of application-specific QoS policies and requirements. A static mapping between application and resource level QoS parameters can then be performed by using HQML QoS Compiler.

We believe that it is not feasible to define end-to-end QoS by just considering one technical domain (e.g., networking, middleware, ASP or applications). Also, SLAs have so far largely been ignored for component execution and middleware in general.

3 Reference model

3.1 Approach

We assume the use of components for assembly of distributed applications or web services and hence assume the use of component oriented middleware. In particular, we concentrate on specific, state-of-the-art application server technologies (J2EE, CORBA Component Model). Of course not every distributed system can be captured this way. For example, streaming systems, such as VIC (Video Conferencing Tool)[1] or RAT (Robust Audio Tool)[2], are ex-

[1] VIC is a video conferencing application developed by the Network Research Group at the Lawrence Berkeley National Laboratory in collaboration with the University of California, Berkeley.

[2] Robust Audio Tool is a an open-source audio conferencing and streaming application by UCL Network and Multimedia Research Group

cluded by this assumption. The class addressed by component middleware is nevertheless extremely important as application server technologies are extensively employed in e-Business and are used to host the components that provide web services.

We associate QoS targets (e.g., performance, availability, reliability, etc.) with identifiable Application Service Provider (ASP) architectural elements, so that the estimation of QoS parameters is informed by the structure of an application's deployment.

Another key point is that QoS semantics of our language do not refer to an ASP model as a whole. They are instead defined according to the diverse domains of the performance properties. For example, the throughput of a database server and the throughput of a component container server are quite different concepts: the former is defined in terms of the query response time varying the number of active connections, the latter in terms of the round-trip method invocations per second. They both contribute to the overall QoS, but one should be able to control each of them separately and in a different way before composing the results.

Similarly, QoS syntax can be very different depending on the reference domain. Performance for an application using a web service is given by, mean completion time for the service (sec), mean peak period latency (sec), successfully completed transactions (%), whereas for an application hosting server using network facilities important parameters include delay(ms), jitter(ms), packet loss(%), and bandwidth(Mbyte/sec).

Ultimately, QoS properties are dependent on the level of abstraction at which the system is being described.

3.2 Service provision reference model

Figure 2 depicts our reference model for a distributed component architecture. The nodes in the model are architectural components. The edges depict opportunities for service level agreements between two parties.

Applications are clients that use either components or web services to deliver end-user services. Web services may be implemented by invoking components. Components provide an abstraction of the underlying resources, enriching their functionalities via middleware support. Containers host component instances and are responsible for managing the underlying resource services for communication, persistence, transactions, security and so forth, and for providing those to components.

In order to make such services QoS enabled, containers need support for QoS negotiation, establishment and monitoring. Container entities are depicted in Figure 2 immediately under component entities. In a business scenario, the

at University College London.

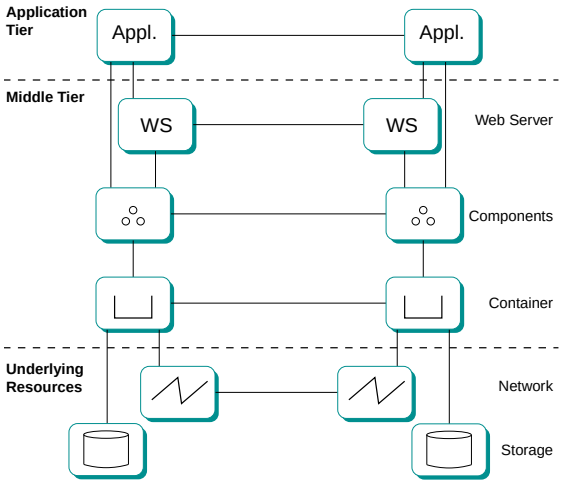

Figure 2. Service provision reference model

role containers are provided by ASPs, called upon to host (other parties') application components.

The underlying resource tier includes network and storage service providers. An ASP can hence interact with Storage Service Providers (SSP) and Internet Service Providers (ISP), and contract specific agreements with them for the provision of services and their related quality.

3.2.1 Vertical and Horizontal SLAs

This architecture facilitates the definition of different levels of abstraction for compiling SLAs. In addition to tier-specific differentiation, we adopt another important SLA classification: Horizontal SLAs govern the interaction between coordinated peers, whereas Vertical SLAs between subordinated pairs, within the service provision architecture stack. Intuitively, they are represented in Figure 2 as horizontal and vertical arcs.

Horizontal SLAs are contracted between different parties providing the same kind of service. For example, two container providers can collaborate for replicating components. *Vertical* SLAs regulate the support parties get from their underlying infrastructure. For example, a container provider can define an agreement with an ISP for network services. Once again, the resulting types of SLA differ in terms of their expressiveness, and SLAng defines them separately.

3.2.2 Crossing organisational boundaries

The SLAng reference model is structured to handle every possible combination of business interactions. Obviously, organisational boundaries can include more than one box in Figure 2, thus resulting in diverse roles included in a single competence domain.

A *Competence Domain* is a non-empty set of abstractions, representing a business party for a particular e-business collaborative process. The party can sign SLA contracts with parties providing other competence domains. Nothing prevents a business party from being represented by several competence domains for different e-business agreements, providing flexibility for business-to-business interaction.

4 SLA definition language (SLAng)

A service level agreement is an arrangement between a customer and a provider, describing technical and non-technical characteristics of a service, including QoS requirements and the related set of metrics with which provision of these requirements is being measured.

In this Section, we analyse the structure of SLAng language. The main requirements we had in mind while developing it were:

Parameterisation Each SLA includes a set of parameters, the values of which quantitatively describe a service. Since they are tier- and actors-specific, a set of parameters of a particular kind of SLA provides a qualitative description of a service.

Compositionality A service can be the result of a cooperation between different domain entities. An SLA language has to enable such composition.

Validation Before initiating an SLA, contractors have to be able to validate it, check its syntax and consistency, further verified as a result of a composition.

Monitoring Ideally, parties should be able to automatically monitor the extent of which the service levels set forth in an agreement are actually provided by its providers.

Enforcement Once service levels are agreed, network routers, database management systems, middleware and web servers can be extended to enforce service levels in an automated manner by using techniques such as caching, replication, clustering and farming.

The SLAng syntax is defined using XML Schema. Using schemas favours the integration with existing service description languages. For example, SLAng can be combined with WSDL and BPEL (all of which are defined using XML schemas) to obtain a complete e-Business automation solution (see Figure 1).

The content of an SLA varies depending on the service offered and incorporates the elements and attributes required for the particular negotiation. In general, it includes:

- An end-point description of the contractors (e.g., information on customer/provider location and facilities)
- Contractual statements (e.g., start date, duration of the agreement, charging clauses)
- Service Level Specification (SLS)s, i.e. the technical QoS description and the associated metrics.

4.1 Kinds of SLA

SLAng defines seven different types of SLA. They regulate the possible agreements between the different types of parties identified in our reference model, i.e. *Application, Web Service, Component, Container, Storage and Network*. *Vertical* and *Horizontal* SLAs can be contracted between pairs of them.

The Vertical SLAs are:

Application: between applications/web-services and components.

Hosting: between container and component providers.

Persistence: between a container provider and an SSP.

Communication: between container and network providers.

The Horizontal SLAs are:

Service: between component and web service providers

Container: between container providers

Networking: between network providers

4.2 Responsibilities

A common characteristic of every SLA is the definition of a relationship of responsibility between a client and a server, including technical annex.

Either in a business-to-customer (B2C) or in business-to-business (B2B) interaction, service provision and use are always involved and, consequently, charges and benefits of the two parties have to be clearly stated. Some of them overlap; in SLAng these are termed *Mutual Responsibilities*.

For each kind of SLA, then, a general structure is defined, including responsibilities of the client of the service (*Client*), responsibilities of the service provider (*Server*) and mutual responsibilities (*Mutual*) to be complied by both of them. This is represented in Figure 3, where, just as an example, the *Hosting* SLA pattern is shown (such a subdivision, anyway, is repeated for every kind of SLA). This set of elements is completed by *Id*, through which we can define service and SLA identifications, alphanumeric values used for reference purpose.

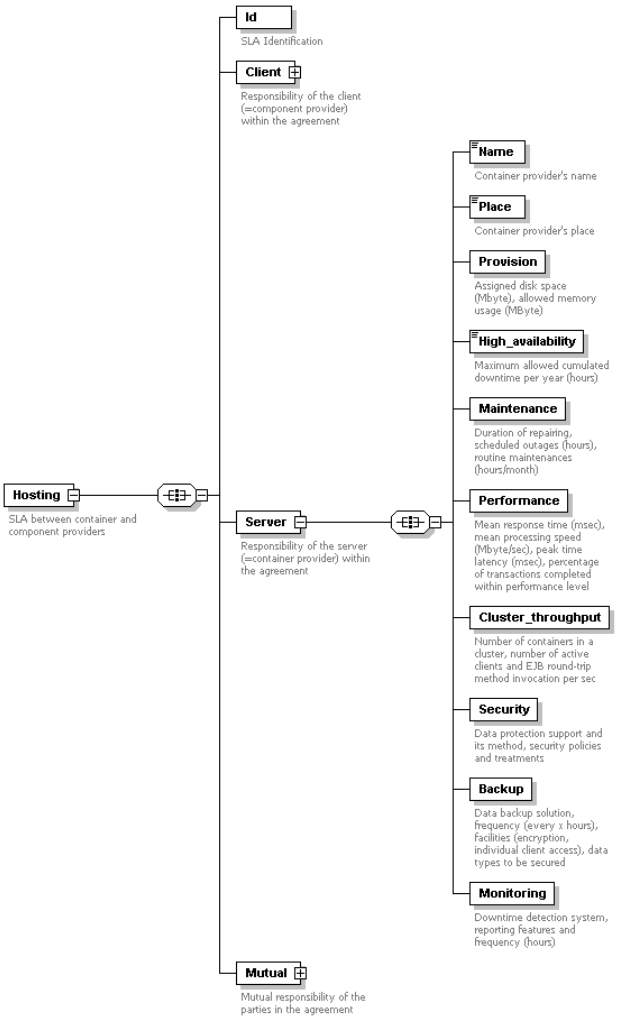

Figure 3. Responsibilities of the server in a *Hosting* SLA

4.3 SLA-specific parameters

Responsibilities are expressed in terms of end-point, contractual and SLS parameters, which are specific to the type of SLA. Such parameters are the leaves of the logical tree representation of SLAng schema.

Figure 3 shows an example with the responsibilities of the service provider (*Server*) in a *Hosting* SLA. An analogous list of SLA parameters is provided for client and mutual responsibilities as well, but they could not be included in the figure for reasons of space.

An analogous pattern is, then, repeated for every kind of SLA. This can give an idea of the dimensions of SLAng schema, but the full detail can not be reported in this paper.

The set of parameters, like the one depicted in Figure 3, is represented by simple-type elements (boxes with a mark in the top left corner), or complex-type elements. The latter are further specified in terms of an element-specific set of attributes.

For example, *Performance* attributes are mean response time in milliseconds, mean processing speed in Megabytes per second, peak time latency in milliseconds, and percentage of transactions completed within a given performance level.

In Section 5, we present an example of SLA that was written in SLAng

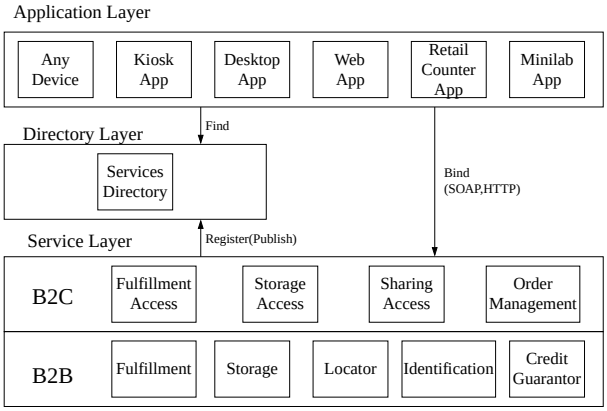

Figure 4. CPXe Architecture

5 Case study: CPXe

The Common Picture eXchange environment (CPXe) is an I3A[3] initiative to develop Internet-based digital photo services. CPXe is an architecture that links digital devices, Internet storage and printing, and retail photo finishing together. It takes advantage of Web Services technologies such as SOAP, WSDL and UDDI and supports a large number of scenarios for imaging applications that are distributed across a number of parties.

As shown in Figure 4, the CPXe architecture enables service providers to define, develop and publish their services, and application providers to look for services and implement interactions with them.

Typical uses of CPXe are enabling print services from home, uploading photos from a kiosk (connecting a digital camera to it) and ordering prints to a particular retailer.

These and other scenarios put in place collaborations that need to be regulated by SLAs, whenever organisational boundaries are crossed.

[3] I3A is the International Imaging Industry Association, setting the standards for the digital imaging markets.

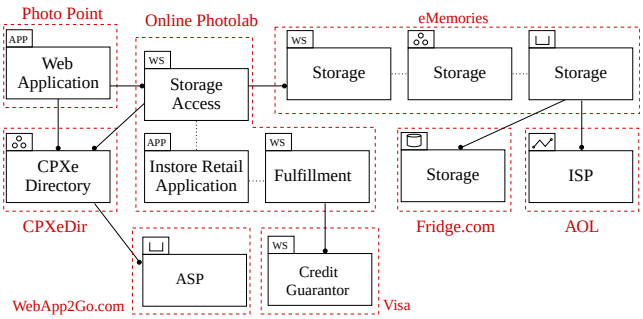

Figure 5. Picture access and printing

5.1 A possible scenario

Figure 5 presents one of four CPXe scenarios that we have analysed in detail. We have used our reference model notation (refer to Section 3.2). SLAs are represented by arcs connecting two entities; a bullet is placed on the *Server* entity side. Entities are differentiated based on their architectural role, stated by an identifier mark over their box.

Competence domains are delimited by dashed line polygons. Every competence domain contains the abstractions that a company is responsible for within an SLA. Relationships between entities in the same competence domain, are represented by a dotted line. Even though they do not have a legal value, they can help the process of internal service-composition modeling and external SLA-offering.

Our case-study scenario intends to show that choosing a business partner can be based on choosing the best service level offer. *Online Photolab* is an application suite intended for use in print shops (Figure 5). The retailer can connect and let their customers connect to a storage service. In this way, customers can store images and eventually retrieve them while in a print shop.

Photo Point, a web application which lets users discover CPXe services, finds *Online Photolab* in *CPXe-Dir*. *Online Photolab* looks up a suitable storage service and choses *e-Memories*, because it ensures an availability rate of 95% and an incremental backup interval of 24 hours. This is stated in a *Service* SLA.

e-Memories relies on *Fridge.com*, a SSP whose guaranteed mean query response-time is 30ms and TTR (Time To Repair) 1 hour. Between them a *Persistence* SLA is stipulated.

Payments can be made using a Web Service provided by *Visa*, which authorizes credit card transactions. *Visa* guarantees a transaction success rate of 99.2% and offers a monitoring report frequency of 12 hours.

CPXe-Directory replication at an ASP, *WebApp2Go.com*, is further represented. Between them there is a *Hosting* SLA. An EJB round-trip method invocation per second of 53ms and availability rate of 99.6% are

```
<?xml version="1.0" encoding="UTF-8"?>
<SLAng xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="dave/TAPAS/SLAng0_5/SLAng0_5.xsd">
 <Vertical>
  <Hosting>
   <Id sls_id="49258" service_id="Replication"/>
   <Client>
    <Name>CPXe-Dir</Name>
    <Place>Los Angeles</Place>
    <Clients mean_number="24000" maximum_number="40000" arrival_rate="113.2"/>
    <Availability>95%</Availability>
   </Client>
   <Server>
    <Name>WebApp2Go.com</Name>
    <Place>London</Place>
    <Provision disk_space="500" memory_usage="512"/>
    <High_availability>99.6%</High_availability>
    <Maintenance recovery_time="2" scheduled_outages="20"
            routine_maintenances="12"/>
    <Performance response_time="2.6" peak_time_latency="4.7"
            successful_transactions="98%" processing_speed="843"/>
    <Cluster_throughput containers="9" active_clients="310"
            method_invocation="53.141"/>
    <Security data_protection="true" encryption_method="RSA"
            certificate="true" user_authentication="true"
            intrusion_detection="false" eavesdrop_prevention="false"/>
    <Backup solution="REOBack" complete_backup_interval="24"
            incremental_backup_interval="6" data_types="User configuration data"
            archiving_form="rar" client_access="true" backup_encryption="false"
            individual_client_backup="true"/>
    <Monitoring tracking_system="IDX System" report_method="XML"
            report_frequency="48" reporting_on_demand="false"/>
   </Server>
   <Mutual>
    <Service_schedule start="2002-12-13" end="2003-12-13"/>
    <Failure_clauses compensation="(100%-availability)*4.6"
            exclusion_clauses="Client caused outages"/>
   </Mutual>
  </Hosting>
 </Vertical>
</SLAng>
```

Figure 6. *Hosting* SLA between a component provider and an ASP

determined in this SLA.

A business relationship with an ISP, *AOL*, is also shown. *AOL* offers 2 Mbps of bandwidth with 0.01% packet loss. The SLA between *AOL* and *Fridge.com* is a *Communication* SLA.

5.2 An SLA for the scenario

While presenting the case-study scenario, we discussed some performance parameters offered by service providers. We cited one or two of them per SLA, the ones considered decisive for choosing a certain provider. In this section, a full SLA of those sketched in Figure 5 is shown to convey the expressiveness of SLAng.

6 Evaluation

Using an industrial case study, we have convinced ourselves that SLAng is expressive enough to represent the QoS parameters required for the complete definition of interfaces in multi-party deployments. We have achieved this by exploiting the different abstractions that we have identified in our reference model and by using abstraction-specific parameters for the necessary interfaces.

We note that the SLAs at these different tiers are precise and having conducted the case study, we can state that

SLAng meets the requirements outlined earlier in this paper. We also note that SLAng allows for fairly concise SLA specifications. There is no service level agreement in the CPXe case study that is longer than 2 KBytes.

We also note that the fact that these SLAs are determined in an XML language has turned out to have a number of advantages. Tools such as XML Spy or ECLIPSE are available to edit and validate SLAs against the language specification. Moreover, we can easily translate SLAs into other representations using XSLT style sheets [3]. We have been able to transform SLAs into a more readable format that is more suitable for inclusion in a service contract. Likewise some of these SLAs could be transformed using XSLT style sheets into deployment descriptors for web servers or application servers.

While conducting the CPXe case study we also noted, however, that further work is necessary on the definition of the semantics of SLAng. Right now, the semantics are defined informally, which has turned out to be a weakness. Instead, it will be necessary to underpin at least some of the definitions, such as latency or throughput of SLA parameters with a more formal semantic model.

7 Conclusion and Further Work

SLAng can specify tier-specific horizontal and vertical SLAs between service users and providers. It is easily extensible to increase expressiveness and combinable with flourishing e-Business automation technologies. It allows engineers to integrate the specification of non-functional features (service levels) of contracts between independent parties with the functional design of a distributed component system for service provisioning.

We will continue to use SLAng to model and reason about SLA composition, analysing its implications. Using an XML-based representation of SLAs provides the possibility of using specialised UML tools for software performance engineering design.

On our agenda there is a study of the benefits of inserting SLAng instances into standard XML-based deployment descriptors, to make components hosting QoS-aware. We also intend to test the effectiveness of SLAng for monitoring compliance to SLAs.

Future work includes also the development of a toolkit for service composition and analysis to assist ASPs in determining what SLAs they can undertake to meet. Model checking techniques could be prove appropriate in this context [11].

References

[1] S. Aissi, P. Malu, and K. Srinivasan. E-business Process Modeling: The Next Big Step. *Computer*, 35(5), May 2002. Innovative technologies for computer professionals.

[2] M. Bearman. ODP-Trader. In *Proc. of the IFIP TC6/WG6.1 Int. Conf. on Open Distributed Processing, Berlin, Germany*, pages 341–352. North-Holland, 1993.

[3] J. Clark. XSL Transformations (XSLT). Technical Report http://www.w3.org/TR/xslt, World Wide Web Consortium, November 1999.

[4] R. Gibbens, S. Sargood, F. Kelly, M. Azmoodeh, R. Macfadyen, and N. Macfadyen. An approach to service level agreements for IP networks with differentiated services. Technical report, Statistical laboratory, University of Cambridge, January 2000.

[5] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML-based Quality of Service Enabling Language for the Web. Technical report, University of Illinois, April 2001.

[6] Y. Krishnamurthy, V. Kachroo, D. A. Karr, C. Rodrigues, J. P. Loyall, R. E. Schantz, and D. C. Schmidt. Integration of QoS-Enabled Distributed Object Computing Middleware for Developing Next-Generation Distributed Applications. In *Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems, Snowboard, Utah*. (OM 2001), June 2001.

[7] J. P. Loyall, R. E. Schantz, J. A. Zinky, and D. E. Bakken. Specifying and Measuring Quality of Service in Distributed Object Systems. In *Proceedings of the First International Symposium on Object-Oriented Real-Time Distributed Computing*. (ISORC '98), April 1998. Kyoto, Japan.

[8] G. Memenios, G. Pavlou, D. Griffin, and L. Georgiadis. Service Level Specification Semantics and Parameters. Internet Draft, tequila-sls-02, February 2002.

[9] Object Management Group. *CORBAservices: Common Object Services Specification, Revised Edition.* 492 Old Connecticut Path, Framingham, MA 01701, USA, December 1998.

[10] S. Seely. *SOAP: Cross Platform Web Service Development Using XML*. Prentice Hall PTR, 2002. ISBN: 0-13-090763-4.

[11] J. Skene and W. Emmerich. Model Driven Performance Analysis of Enterprise Computing Systems. Research note, UCL Dept. of Computer Science, December 2002. Submitted for publication.