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 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 in a quality of service aware manner. We evaluate SLAng based on the experience gained from this case study.

1 Introduction

The term 'electronic business', or 'e-business', refers to the execution of transactions of commercial significance in a distributed computing context. Such transactions may involve an organisation, its clients and its partners, or they may be internal to an organisation, integrating separate computational assets.

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.

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

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 Quality of Service (QoS) properties.

The novel contribution of this paper is a reference model for inter-organisational service provision at storage, network, middleware and application level. The model provides the basis for the definition of SLAng, a language for Service Level Agreements (SLAs). SLAs capture the mutual responsibilities of the provider of a service and its client with respect to non-functional properties. Our language 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 evaluated the expressiveness of SLAng using a case study that supports the QoS-aware implementation of web services for image processing.

The remainder of this paper is structured as follows: In Section 2 we review related work; in Section 3 we discuss our approach and present our reference model for inter-organisational service provision and use; in Section 4, we present SLAng in detail; in Section 5 we describe a case study in which SLAng was used to capture QoS responsibilities shared by components of the CPXe application; Section 6 critically evaluates SLAng and in Section 7, we summarise our contributions and discuss future directions for our research.

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

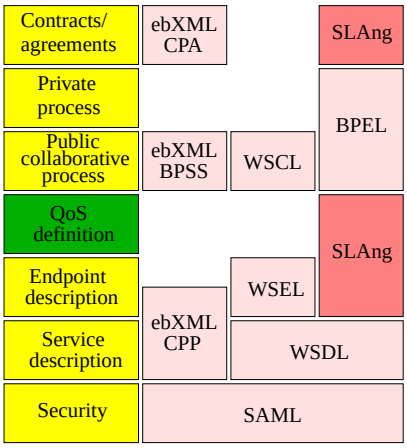

Fig. 1. e-Business automation standards

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

In this section, we outline the underlying assumptions for our research and then present a reference model that provides the basis for our service level agreement language SLAng.

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 excluded 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

[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 at University College London.

service level agreements between two parties. The duplication of the traditional layered architecture reflects our observation that service provision can occur at any level in the architecture. Moreover, the model takes distributed deployment to the extreme in that it assumes that different parties can be involved in using, developing and hosting components as well as providing underlying resources, such as network connectivity and storage for deployment.

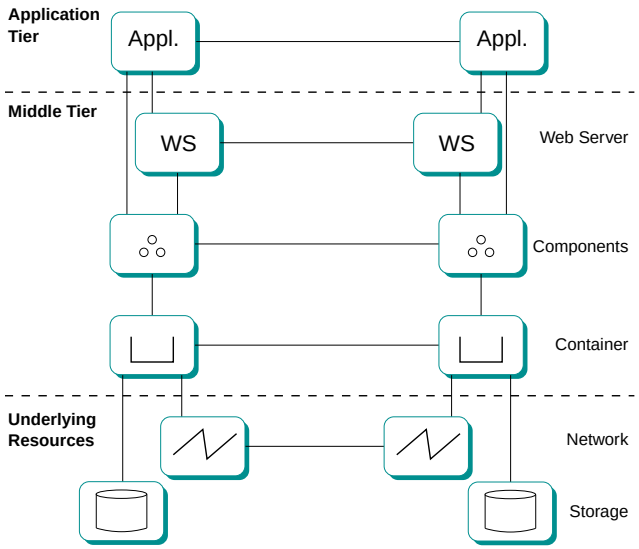

Fig. 2. Service provision reference model

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 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.

The architectural components depicted in the figure can each be owned by a seperate organisation. Hence, each box can also represent a party of a bilateral business agreement. For each possible SLA, we define a particular template, so that different

QoS parameters and attributes can be specified for every tier-specific and party-specific SLA.

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.

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.

The first goal of an SLA definition language is to provide the capability to express, with the maximum degree of accuracy, the qualitative and quantitative features of a service. Through such a language, e-business parties can rapidly and precisely formulate the level of a service while offering it. Furthermore, it is convenient to refer to a standard, which everyone is able to use and understand.

Other relevant achievements are the possibility to easily make comparisons between offers, to advertise and retrieve information about them, to reason about service proposals, understanding what one can offer and expect to receive, and to easily monitor QoS guarantees, both for fulfilling and claiming them.

The main requirements for achieving these goals we had in mind while developing SLAng were parameterisation, compositionality, validation, monitoring and enforcement:

Parameterisation Each SLA includes a set of parameters, the values of which quantitatively describe a service. For what we have stated previously, they have to be tier- and actors-specific; hence, a set of parameters of a particular kind of SLA provides a qualitative description of a service.

Compositionality In a multi-domain environment, a service can be the result of a cooperation between different domain entities. Services can be cascaded or aggregated and, hence, service providers should be able to compose SLAs in order to issue new offers to customers. 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. Furthermore, validity should be 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. Likewise, a party should be able to deduce the extent with which it has met the service levels it agreed to provide to its customers. SLAs should therefore provide the basis for the derivation and installation of automated monitors that report extents with which service levels are being met.

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.

4.1 SLAng key concepts

SLAng is an XML language for capturing Service Level Agreements. In order to be legally binding, an SLA has to be embedded in what is called SLA contract, i.e. a framework containing one or more SLAs plus the names of the two juridical persons contracting the agreement and possibly of a trusted third party, together with their digital signatures (Figure 3).

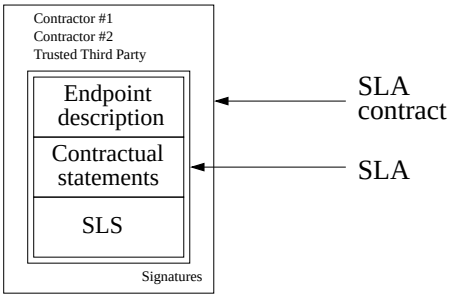

Fig. 3. SLA contract structure

XML proves ideal for parameterisation. Parameterisation of service level specifications is supported at different system tiers, including vertical and horizontal agreements.

In order to allow compositionality, we focus on interfaces. A service can be seen as the result of performing a set of operations. An interface is a set of points of interactions, through which the functionality of such operations can be accessed/provided and, hence, it is located at the logical boundary between the user and provider entities/systems. Our effort is to add access and provision of non-functional characteristics of service to service delivery interfaces.

Each party identified in our reference model is responsible only for its interfaces with other parties, and the guarantees it assures are the outcome of the composition of the guarantees it expects from other interfaces with parties it has agreements with. A failure-clause mechanism regulates such a composition, so that compensation is the result of cascading responsibilities.

In Section 3.2, we stated that an SLA contract can be signed by two competence domains. Nothing prevents an organisation, though, from using SLAng within its own boundaries and producing an internal chain of SLAs. Even if legal implications are not relevant in this case, SLAng can still help the process of internal service-composition modeling, so that an accurate SLA proposal can be offered to external business parties. This can be very useful for widely distributed organisations which are, e.g, component and container providers at the same time and want to use their own database facilities. Moreover, once the service level is precisely determined, SLAng instances can be inserted or transformed into standard component deployment descriptors, while deploying the service components.

4.2 SLAng structure

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). Moreover, one can take advantage of a variety of existing XML tools and parsers. In this section, we analyse the structure of our language.

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, rebates on SLA violation)
- Service Level Specification (SLS)s, i.e. the technical QoS description and the associated metrics.

The latter is the challenging issue, because, as we said, performance guarantees are difficult to precisely determine and maintain. They include availability, response time, utilisation and other tier-specific QoS targets, all of which are the result of several dependencies of a multi-domain service provision scenario.

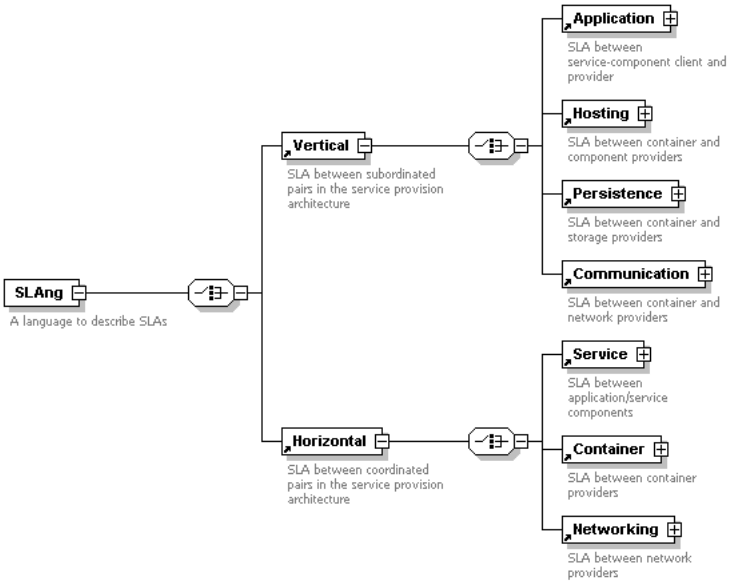

Fig. 4. Vertical and Horizontal SLAs

Kinds of SLA SLAng defines seven different types of SLA, four of which are vertical and three horizontal. 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.*

Figure 4 shows the name of SLAs that can be contracted between pairs of them, appropriately subdivided into *Vertical* and *Horizontal* agreements. The Vertical SLAs are:

Application: between applications or web services and components,
Hosting: between container and component providers,
Persistence: between a container provider and an SSP, and
Communication: between container and network service providers.

The Horizontal SLAs that parties enter into by composing vertical SLAs are:

Service: between component and web service providers
Container: between container providers
Networking: between network providers

Responsibilities A common characteristic of every SLA is the definition of a relationship of mutual responsibility between a client and a server, including technical annex. Since they are bilateral agreements, a description of service client and server responsibilities is needed.

Either in a business-to-customer or in business-to-business 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 5, where, just as an example, the *Communication* 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.

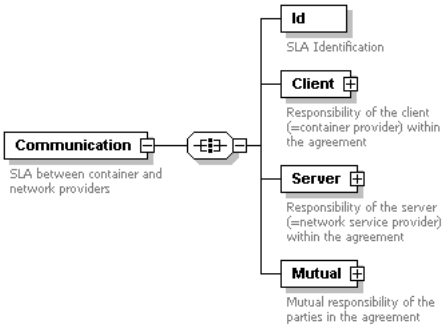

Fig. 5. General structure for an SLA

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 6 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. We believe that there is much more significance to present the underlying concepts of our research, leaving a detailed understanding of all the features provided by SLAng to the free examination of its XML code.

The set of parameters, like the one depicted in Figure 6, 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 several examples of SLAs that were written in SLAng to express QoS concerns of different parties involved in our case study.

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 7, 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. They both make use of well-defined CPXe interfaces and can additionally provide/use Service Locators to apply business rules, consumer information and service properties to filter the list of services in the CPXe directory.

CPXe Applications can be categorized into in-home, on-line and in-store applications. They support Internet-enabled devices (e.g., cameras, phones, PDAs), kiosks, desktop software, web applications and behind the counter applications, including minilab applications.

Services are subdived into business-to-customer (B2C) and business-to-business (B2B) services, where the former provide customers for access to the latter. The CPXe Directory provides information about the owner of the service, its interface definition and its location on the network. CPXe-service providers specify hardware, network and software service definition entries, based on a CPXe service API template.

Several uses of CPXe can be accomplished. A typical one is enabling print services from home, locating, via a desktop application, a fulfillment service, that manages the order and send it to the nearest (or favourite) retail shop. Picture access from a retailer application is also possible, using an in-store application that retrieves pictures from an on-line storage and sets the fulfillment service proceeding. Also, one could upload photos from a kiosk (connecting a digital camera to it) and order prints to a particular retailer or even have them mailed to home.

5.1 Aggregation of services: implications

In order to appear to service requesters as having more capabilities than autonomously provided, CPXe parties have to collaborate. This section highlights the issues concerning multi-party cooperation which are relevant for our case study, and shows how SLAng can provide SLA negotiation support to CPXe. This way QoS can be delivered together with the service, both of them as a result of the business collaboration.

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

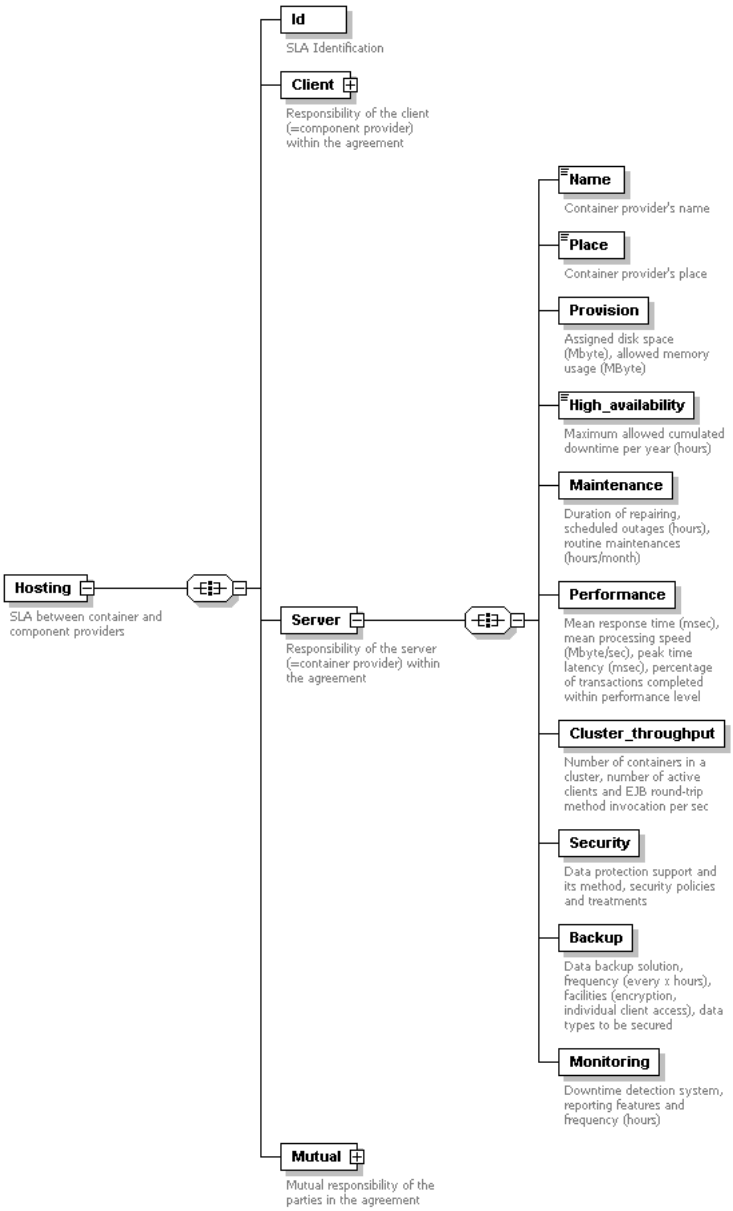

Fig. 6. Responsibilities of the server in a *Hosting* SLA

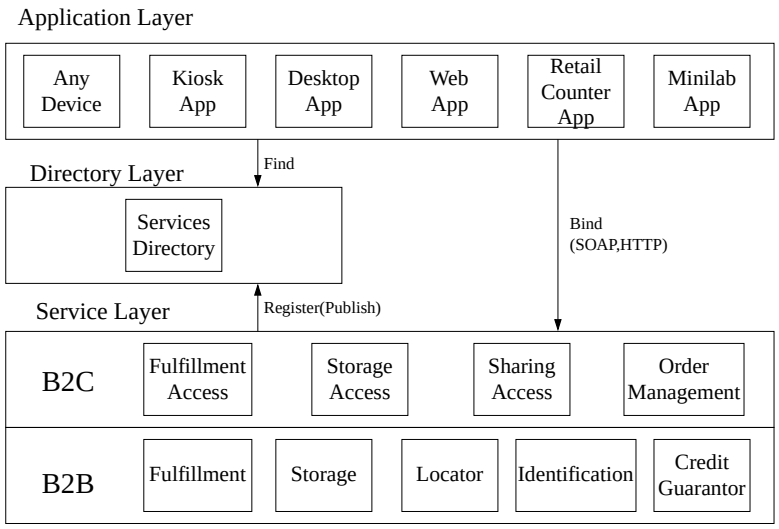

Fig. 7. CPXe Architecture

CPXe applications can either be totally integrated using web services, thus providing total consumer interaction, or stand alone. In the former case, they can further be augmented by other CPXe applications. Applications can, then, locate services by either interacting with CPXe directory itself, or by using another service, a smart locator, that applies business specific filters.

These and other scenarios put in place collaborations that need to be regulated by SLAs, whenever organisational boundaries are crossed. CPXe entities, then, can hold different roles and possibly change them with respect to their partners. The bilateral and compositional nature of SLAng SLAs supports this naturally.

At the B2C interface, access services to storage, fulfillment and sharing are provided. The associated applications are invoked by a consumer (e.g., via a browser) or by another application (e.g., desktop or kiosk applications). An order management service can be provided to them as well.

Fulfillment services are logically located at B2B interfaces, where properties of a service can be set and orders can be placed, canceled, modified. Images can be pushed from the requester, pulled from an on-line service or referenced in a CPXe-compliant storage service.

Storage services are provided at B2B interface, as well. Digital media can be uploaded from customers (e.g., using digital cameras/scanner from home or from a kiosk) or they can be digitalised and uploaded from minilabs equipment.

5.2 A possible scenario

Figure 8 presents one of four CPXe scenarios that we have analysed in detail. In order to describe service composition, we have used our reference model notation (refer to

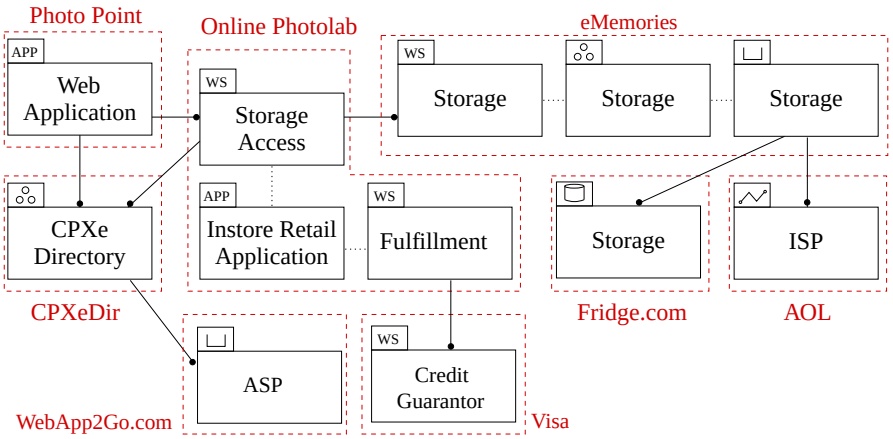

Fig. 8. Picture access and printing

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; the name of the company is put near its competence domain. Relationships between entities in the same competence domain, are represented by a dotted line. We repeat the statement that SLAng-generated SLAs within the same business boundaries can help the process of internal service-composition modeling and external SLA-offering (Section 4.1). SLAs with a legal value (SLA contracts) are those that cross domain boundaries.

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 8). 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, deciding to print a selection of them.

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 that will be used both by the customer and by the retailer. In this case *e-Memories* is choosen, because it ensures an availability rate of 95% and an incremental backup interval of 24 hours. This is stated in a *Service* SLA.

Figure 8 provides a detailed description of the *e-Memories Competence Domain*, including internal relationships between Web Services, Components and Container. Elsewhere, e.g. for the credit guarantor, only *Web Service* entities are depicted within a competence domain. This can mean that the underlying infrastructure for service provisioning belongs to the same business or, even simpler, that we do not represent other possible business interactions, regulated by more SLAs. For example, no *Networking* SLAs are illustrated, even if they are likely to be in place. Service composition, indeed,

allows us to focus only on service interfaces we are interested in and consider the others as given.

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 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.

5.3 SLAs for the scenario

While presenting the case-study scenario, we discussed some performance parameters offered by service providers. In each SLA there are several parameters and, just for illustrative purpose, we cited one or two of them per SLA, the ones considered decisive for choosing a certain provider. In this section, four full SLAs of those sketched in Figure 8 are shown to convey the expressiveness of SLAng.

```
<?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">
 <Horizontal>
  <Service>
   <Id sls_id="453A" service_id="storage"/>
   <Client>
    <Name>Online Photolab</Name>
    <Place>Los Angeles</Place>
    <Availability>96%</Availability>
   </Client>
   <Server>
    <Name>e-Memories</Name>
    <Place>Paris</Place>
    <Availability>95%</Availability>
    <Maintenance recovery_time="2" scheduled_outages="8" routine_maintenances="5"/>
    <Backup solution="Softbackup" complete_backup_interval="72"
            incremental_backup_interval="24" data_types="Log data" archiving_form="zip"
            client_access="true" backup_encryption="true" individual_client_backup="true"/>
   </Server>
   <Mutual>
    <Service_schedule start="2002-12-13" end="2010-12-13"/>
    <Performance>
     <Service_time average="7.3" maximum="16.9" minimum="4.6"/>
     <Service_rate>26.7</Service_rate>
    </Performance>
    <Clients>3056</Clients>
    <Security data_protection="true" encryption_method="RSA" certificate="true"
             user_authentication="true" intrusion_detection="true" virus_scanning="true"
             eavesdrop_prevention="true"/>
    <Monitoring tracking_system="EHS Performance Tracking" report_method="XML"
                report_frequency="24" reporting_on_demand="true" security_violations="true"/>
    <Failure_clauses compensation="(100%-availability)*0.8"
                     exclusion_clauses="routine maintenances"/>
   </Mutual>
  </Service>
 </Horizontal>
</SLAng>
```

Fig. 9. *Service* SLA between two Web Services

```
<?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>
  <Persistence>
   <Id sls_id="fgh812g" service_id="image_storing"/>
   <Client>
    <Name>e-Memories</Name>
    <Place>Paris</Place>
    <Users mean_number="18400" maximum_number="35000" arrival_rate="174.4"/>
    <Availability>99.6%</Availability>
   </Client>
   <Server>
    <Name>Fridge.com</Name>
    <Place>Houston, Texas</Place>
    <Provision disk_space="400"/>
    <Availability>97%</Availability>
    <Reliability>90%</Reliability>
    <Maintenance recovery_time="1" scheduled_outages="17" routine_maintenances="24"/>
    <Query_response_time average="30" maximum="48" minimum="21"/>
    <Data_integrity>97%</Data_integrity>
    <Security encrypted_storage="true" encryption_method="DES" certificate="false"
              user_authentication="true" intrusion_detection="true" virus_scanning="false"
              eavesdrop_prevention="true"/>
    <Backup solution="CommVault" complete_backup_interval="48"
            incremental_backup_interval="12" data_types="all" archiving_form="tar"
            client_access="true" backup_encryption="true" individual_client_backup="true"/>
    <Monitoring tracking_system="RTS" report_method="XML" report_frequency="72"
                reporting_on_demand="false" security_violations="false"/>
   </Server>
   <Mutual>
    <Service_schedule start="2002-12-13" end="2005-12-13"/>
    <Cpu_utilisation>75%</Cpu_utilisation>
    <Memory_usage>80%</Memory_usage>
    <Connection_entries>4967295</Connection_entries>
    <Users>4294964225</Users>
    <Failure_clauses compensation="(100%-availability)*2.8"
                     exclusion_clauses="routine maintenances"/>
   </Mutual>
  </Persistence>
 </Vertical>
</SLAng>
```

Fig. 10. *Persistence* SLA between a container provider and an SSP

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.

```
<?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"
              virus_scanning="true" 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" security_violations="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>
```

Fig. 11. *Hosting* SLA between a component provider and an ASP

```
<?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>
   <Communication>
    <Id sls_id="357BN" service_id="FracT3"/>
    <Client>
     <Name>e-Memories</Name>
     <Place>Paris</Place>
     <Usage arrival_rate="14.2" MTU="1024" access_link_limitation="50%"/>
     <Availability>99.6%</Availability>
    </Client>
    <Server>
     <Name>AOL</Name>
     <Place>Connecticut</Place>
     <Maintenance recovery_time="1" scheduled_outages="6" routine_maintenances="12"/>
     <Reliability downtime="20"/>
     <Performance_guarantees delay="2.3" jitter="0.7" packet_loss="0.01%" bandwidth="2M"/>
     <Security firewall="true" firewall_system="NetBSD" user_authentication="true"
               intrusion_detection="true" ipsec="true" eavesdrop_prevention="false"/>
     <DiffServ_level>010100</DiffServ_level>
     <Monitoring tracking_system="EHS Performance Tracking" report_method="XML Document"
                 report_frequency="3/week" reporting_on_demand="true" security_violations="true"/>
    </Server>
    <Mutual>
     <Service_schedule start="2002-12-13" end="2004-12-13"/>
     <Failure_clauses compensation="(100%-availability)*1.2"
                      exclusion_clauses="Client caused outages"/>
    </Mutual>
   </Communication>
 </Vertical>
</SLAng>
```

Fig. 12. *Communication* SLA between container provider and an ISP

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].

Acknowledgements

We would like to thank Jon Crowcroft, Fabio Panzieri, Nicola Mezzetti, Werner Beckmann and Santosh Shrivastava together with all TAPAS partners for the fruitful discussion that helped us to refine SLAng. We are indebted to Karen Lawson for drawing our attention to CPXe.

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. J. Gibbens, S. K. Sargood, F. P. 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, Department of Computer Science, 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.