# D2: SPECIFICATION LANGUAGE FOR SERVICE LEVEL AGREEMENTS

DOCUMENT SUBMITTED TO

EU IST PROJECT 34069

TAPAS.

By

Domenico Davide Lamanna, James Skene and Wolfgang Emmerich

University College London

March 2003

# Contents

# List of Figures

# Preface

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.

Unfortunately, combining functionality is not the only requirement for e-business integration. Non-functional or 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. Our work stands in the context of these efforts, but addresses the need for description and negotiation of Quality of Service (QoS) properties.

QoS has considerably been investigated for network protocols and services. We take some of those ideas to the middleware and application level. To do so, we introduce 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.

## Organization of the manuscript

This document is further structured as follows. The first chapter looks at the big picture of e-Business automation. It provides a survey of the evolution of e-Business and describes the overall thread between the new technologies which have been proposed. The solutions choosen by the TAPAS project are, this way, set into the scene.

Chapter 2 presents the middleware architecture SLAng is based on, for the provision of application services. In this chapter we first present middleware issues for the execution of distributed processes. We then move on to discuss the concepts of SLAng.

Chapter 3 introduces SLAng features. Its basic structure and key concepts are outlined, in order to set the framework for Chapter 4, where we illustrate in

details its constructs. There we present the QoS parameters that govern service components deployment, explaining how they are organised in SLAng language.

Provided the possibility to specify non-functional aspects while deploying service components, Chapter 5 discuss a case study we have been testing SLAng efficacy on.

An overall evaluation of the work is given in Chapter 6 together with a presentation of our research agenda.

# Chapter 1

# E-Business automation: Setting the Scene

## 1.1 Introduction

Internet Electronic Business introduced such innovations in the way of conducting businesses, that it can unquestionably be considered a revolution affecting society and economy, besides a technical challenge. The tremendous speed of its growth favoured the proliferation of several architectures, standards and technologies, out of a coordinated development and, hence, overlooking interoperability issues. In fact, enabling the collaboration between heterogeneous businesses and systems is a crucial point to achieve maximum e-business integration and composition.

The key issue is gathering around the same table, inter-governmental organizations, and sector and industry associations with the objective of actively encouraging organizations to contribute and develop recommendations and standards.

Many efforts have been carried out by international organisations such as Commerce.net, aiming to unify methods for resolving cross-border commercial disputes and to make the Internet a real global market place. Commerce.net promotes initiatives and brings about collaborations among industry partners and government institutions in order to give a reasoned incentive to e-business. Weighty achievements were the involvment of Next Generation Internet (NGI) Program for the

definition of a Global Framework for e-Commerce, and the promotions of a series of conference ("B2B Big Bang" conferences) to address Internet Business-to-Business issues.

In this scenario, the role of World Wide Web consortium (W3C) was crucial in receiving and coordinating inputs from industrial interlocutors such as Ariba, IBM and Microsoft. Specifically, W3C Web Services Activity group defined the concept of Web Services (January 2002), interfaces that describe "a collection of operations that are network accessible through standardized XML messaging" (official W3c definition).

In plain words, e-Business is basically about conducting transactions over the Internet (public collaborative process) and integrating front- and back- office processes (private collaborative process). In order to support service provision and use at the interface between the two entities involved in the trasaction, applications including capabilities to enable such interactions have to be provided. Consequently, e-Business automation and business-to-business requirements accelerated the tendency to use the Internet for application to application communication. At the same time, the advent of XML favoured the exchanging of information between heterogenuous systems in different environments. Web Services are basically the result of these two factors. They can be seen as programmable application components, accessible by other applications via standard Web XML-based protocols.

## 1.2 Industry scene

Research on this field led to the submissiom to W3C of WSDL, Web Services description language, an XML-based specification describing Web Services as sets of message-enabled operations. WSDL provides support for descriptions of interfaces to services and their (network) protocol bindings, but leaves open issues such as security, transaction management and quality of service (QoS), which are currently work in progress.

A substantial amount of research has been carried out on this side. UN/CEFACT

(the United Nations Centre for Trade Facilitation and Electronic Business) and OASIS (Organization for the Advancement of Structured Information Standards) are sponsoring the ebXML specifications, which enable enterprises cooperation by providing support for collaborative processes. Businesses can be conducted through the exchange of XML messages based on a set of choreographed transactions. Details of transport, signaling, security and bindings to a business process can be specified as well [1].

Another relevant initiative of this kind is RosettaNet, a consortium of major Information Technology, Electronic Components and Semiconductor Manufactoring companies. RosettaNet promotes an open e-business process standard to improve operational efficiency via enabling flexible trading networks. Partner interface processes (PIP), i.e. system-to-system XML-based dialogs, are defined, including manufacturers, distributors, resellers, shippers, end users, which are the actors of the business-to-business interactions.

Also IBM is contributing to e-Business automation. They released WSFL (Web Services Flow Language) an XML-based specification for describing Web Services composition to support public collaborative processes. It considers two types of composition: A flow model specifies the logic of a business process in terms of the sequence of functions to be executed; A global model describes the interactions of Web Services involved in the flow and the way partecipants mutually use them in a business process.

Microsoft's XLANG, a framework for private business process, and WSFL, have converged together in BPEL (Business Process Execution Language) for a complete and integrated business process orchestration.

In general, 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.

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

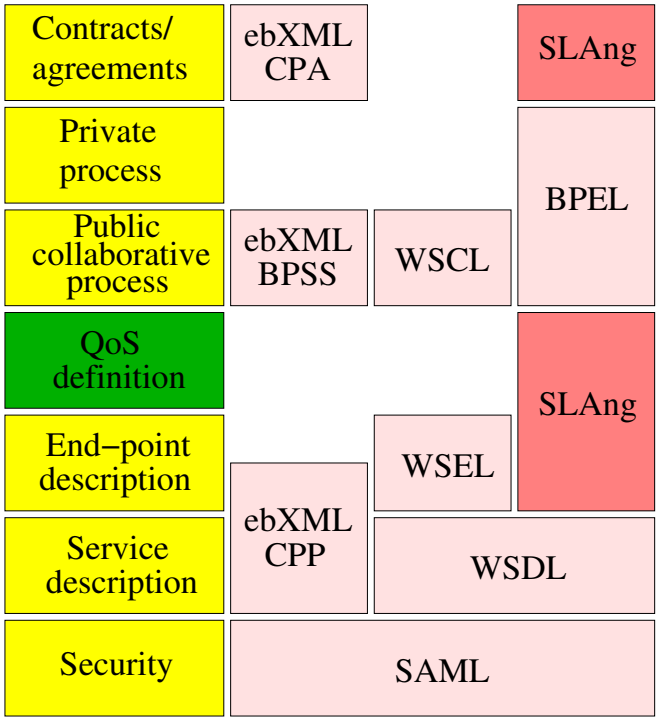



Figure 1: e-Business automation standards

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.

Web Services provide interfaces to software components using standard Internet technologies and make them interact dynamically, thus bridging systems with one another. Operating in such a multidomain scenario led to a number of interoperability issues, i.e. messaging (including typing, operation invoking and protocol binding), need for a registry for service publishing and discovery, security, transactions and QoS.

The former two were addressed by WSDL, SOAP [27] and UDDI (Universal description, discovery and integration). Relevant work on security has been done by OASIS Security Services Technical Committee (SSTC), that recently released SAML (Security Assertion Markup Language), "an XML-based framework for Web services that allows the exchange of authentication and authorization information among business partners".

A substantial amount of work has been carried out, then, on transaction management. Besides the already cited initiatives, Hewlett-Packard developed WSCL (Web Services Conversation Language), XML-based specification to define transitions from one interaction to another, between service providers and consumers.

Very few has been done or even said about QoS in e-Business automation, despite the differentiation, based on it, required by the competive nature of B2B environment. Hence, a need for an additional (and optional) layer within Web Services specification is getting pressing.

IBM's WSEL (Web Services End-point language) goes towards this direction. An end-point provides more than a WSDL port type, since QoS and cost characteristic can be specified while composing and coreographing services into larger business. Unfortunately it is only a project outline in its embryonic stage and further steps are very slow. In fact, adding the complexity of this layer without having a QoS-controlled underlying system architecture is not very helpful.

Also EbXML provides a little support for end-point description in terms of the IT capabilities of a party. This is included in its CPP (Collaborative Protocol Profile), but can hardly be considered a QoS characteristic description. EbXML CPA (Collaborative Protocol Agreement), then, describes the capabilities that two parties have agreed to use, but it calculates them as an intersection of the two CPPs of their, which is of course restrictive and, again, far from exhaustive. Moreover, the integration between Contracts/agreement layer and End-point description layer is loose, that is a deficiency, considering that a Service Level Agreement is a technical contract containing QoS characteristics that are meant to be provided. A SLA is a result of a negotiation; maximum control should be given to contract compilation process in terms of end-point description (and, ideally, of QoS characteristics definition, if any). CPPA technical committe has started to work on it, but nothing was released so far.

SLAng is an answer to all these necessities. It can express end-point description (e.g., service location, provider information, etc.), contract statements (e.g., service duration, failure compensation, etc.) and Service Level Specification (e.g.,

response time, availability, etc.). In plain words, it is really able to generate full Service Level Agreement, in a standard, XML-based form.

The delivery of new end-user services requires a significant development in the service provision architecture. In order to model and determine QoS, SLAs need to be provided at the various levels of abstraction of such an architecture, a feature that is currently missing and that we believe is essential to properly handle such a issue. Adequate level of details about this will be given in Chapter 2. For the moment, we just anticipate that SLAng is multidomain- and multilayer-enabled.

## 1.3 Research scene

The ISO/ODP trading function [2] and its various incarnations, for example the CORBA Trading Service [24] provide for quality of service definition. However, such traders define QoS at a single level of abstraction, finding objects that meet particular QoS. Conversely, we allow for QoS definitions and integration 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 [22]. 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 [13]. 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 [22] 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, application level 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 [19, 16]. 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) [14], 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.

## 1.4 What is SLAng about?

As we said, very few has been done about QoS guarantees and their negotiation. It is clear that such issues represent a key point in a wide, open and competitive market, as the global e-Market is going to be. Service providers which will be able to offer well defined quality standards will certainly prevail. In a multidomain scenario, where different parties collaborate, an agreement on service level between

cooperating organisations is essential to meet quality requirements and deliver quality services to customers.

Service Level Agreement (SLA)s are typically used to sanction such an agreement. They are contracts stating server and client responsibilities and expectations, including legal and technical aspects. They are written in plain English (or in other natural languages) and, even if templates exist which define common fields and related semantics, they are far away from being standardised and systematic, thus resulting in SLAs to be practically written every time all over again. Moreover, since a number of different frameworks to compile them are widespread, comparisons and negotiations are arduous because of their diversity.

At present, little support is available for SLA compilation and negotiation. SLAng, SLA notation generator, is the XML-based language we defined to systematize their treatment. Using SLAng it is possible to produce Service Level Agreements between two parties at (application level, as well as at application services and system resources level), through expressing quality parameters pertaining the particular working level and type of agreement.

## 1.5 Novel contributions

The major contribution of SLAng is filling the gap of QoS definition left by e-Business automation research, by producing a formal language, with a well defined syntax and semantics, for describing service level specifications. The work carried out in this field so far, is focused on application level, where is very difficult to state service levels that can be ensured. A full knowledge and management of system architectures is essential to assess quality capabilities with a sufficient degree of confidentiality. In other words, current research limits itself to endpoint description and very few can be said about actual service availability and responsiveness.

The approach of SLAng, instead, is to deal with service level specification at different levels (application, application services, system resources), and then

composing individual agreements to state the overall quality offer at the desired level. Such an offer can, hence, be much more precise, single responsibilities can be traced and service level guarantees met. Focusing at each level makes quality assesment more effective, because level-specific quality parameters can be tested and expressed.

Moreover, SLAng allows specification of non-functional features (service level) of contracts between independent parties to be integrated with the functional design of a distribuited component system, thus facilitating software performance engineering process and quality assesment.

# Chapter 2

# QoS-aware middleware architecture

## 2.1 Introduction

The increasing use of distributed systems for co-operative execution of software components residing on different hosts, made software engineering more challenging. In order for the distributed system to appear as an integrated computing facility, the components have to cope with network latency, heterogeneity of hardware and software platforms, marshalling of data structures exchanged between components and synchronization of their communication. Layering middleware between network operating systems and application components is the most common and effective approach to such problems.

Current middleware products or implementations, such as the Java 2 Application Servers or CORBA Component Model, focus on the specification of functionality, but do not yet support the expression of Quality of Service (QoS) requirements. Determining scalability and performance requirements, for example, while integrating components in a distributed system, can be very difficult without applying systematic techniques, which are at present not available in a standard and comprehensive framework for software performance engineering.

In order to avoid guesswork, notations and methods are needed that focus on

non-functional requirements and help software engineers to systematically model architectures and systems that will meet a set of Quality of Service (QoS) features. The key issue is to provide quantitative requirements models for the required response time, peak loads and transactions volume that an architecture has to scale up to, besides validation techniques (such as model checking) to validate that models do actually meet the requirements.

The tendency to extend the distribution from a LAN environment towards the Internet, accelerated by the introduction of Web Services technologies, made such a need even more urgent and raised new issues regarding provision of services in a distributed environment.

In this chapter, 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, thus showing how we address these issues.

## 2.2 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

[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.

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.

## 2.3 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. 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.

Applications are clients that use either components or web services to deliver

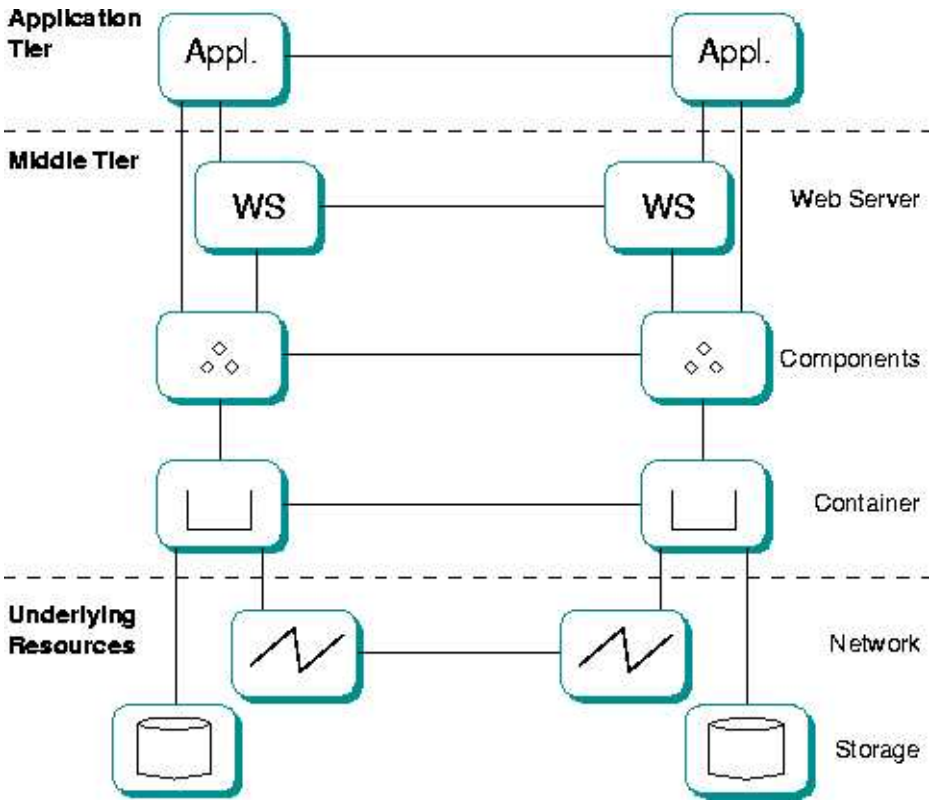



Figure 2: Service provision reference model

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.

### 2.3.1 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.

### 2.3.2 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.

# Chapter 3

# SLAng: a language for defining SLAs

## 3.1 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 [18].

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.

## 3.2 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,

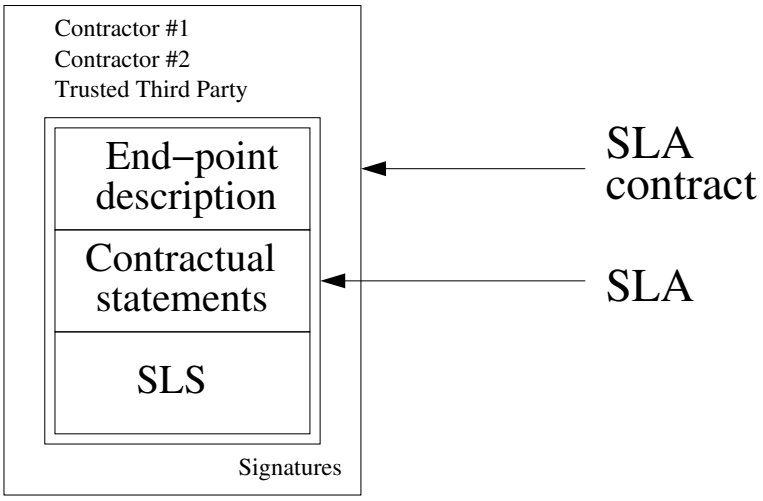



Figure 3: SLA contract structure

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).

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 2.3.2, 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.

## 3.3 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,

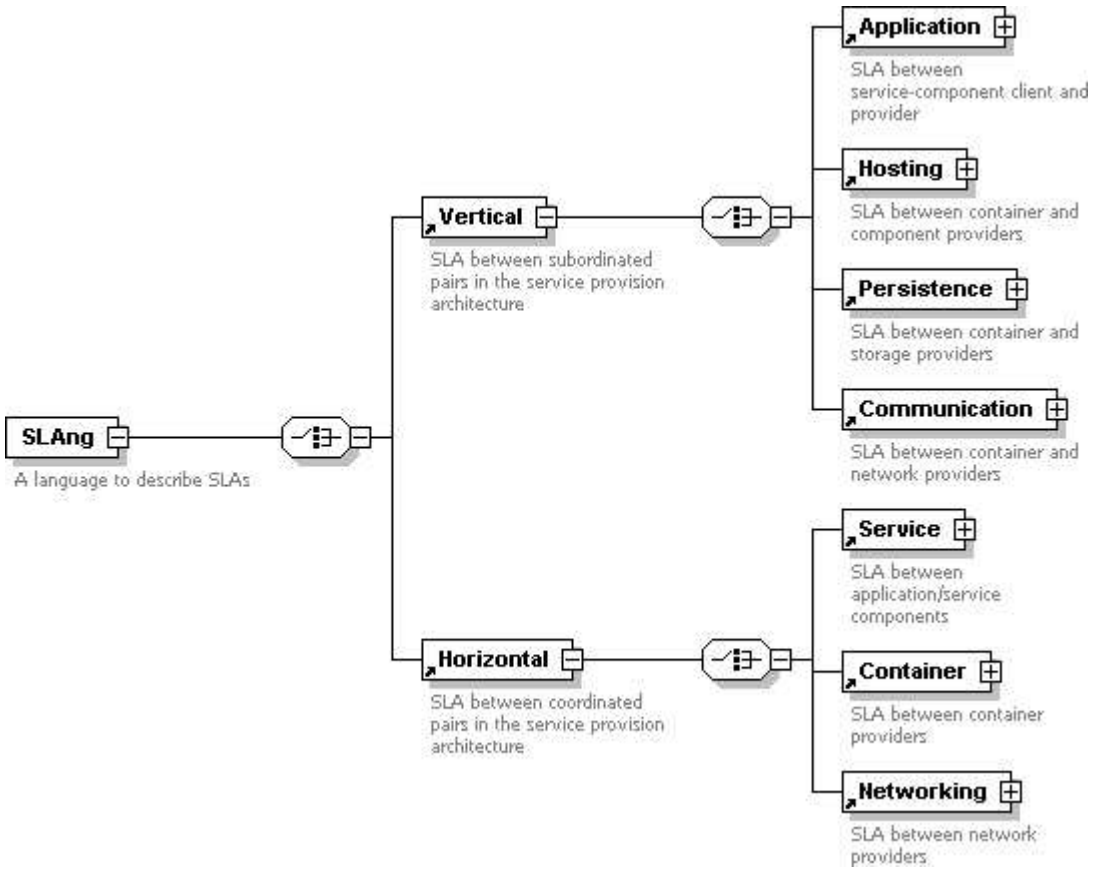



Figure 4: Vertical and Horizontal SLAs

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.

### 3.3.1 Kinds of SLAs

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. This diagram, and the following similar ones, use the W3C schema notation. Square boxes represent schema elements and hexagonal boxes indicate the opportunity

for a choice among sub-elements.

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

### 3.3.2 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.

One can see an example of this in Figure 5 (Chapter 4), where the *Networking* SLA pattern is shown. Such a subdivision, anyway, is repeated for every kind of SLA (we will see this more in details in Chapter 4). This set of elements is

completed by *Id*, through which we can define service and SLA identifications, alphanumeric values used for reference purpose.

# Chapter 4

# SLAng constructs

## 4.1 SLA-specific parameters

As we said in Chapter 3, 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.

A list of SLA parameters is provided for server, client and mutual responsibilities for each SLA we classified. In this Section, we analyse SLAng structure in details, discussing every contruct we defined. We start from *Networking* SLA and we proceed up the stack to *Application* SLA.

We, therefore, outline the definition of SLA parameters and the semantics that go behind this representation in the context of service provision, presenting in parallel the XML schema definition for each of them, thus inspecting step by step the whole body of SLAng.

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 guarantees* attributes are delay, jitter, loss and throughtput. A shaded line border represent an optional element (contractors can choose whether to specify it or not). Hexagonal boxes in this case indicate that all of the elements, unless they are optional, have to be specified.

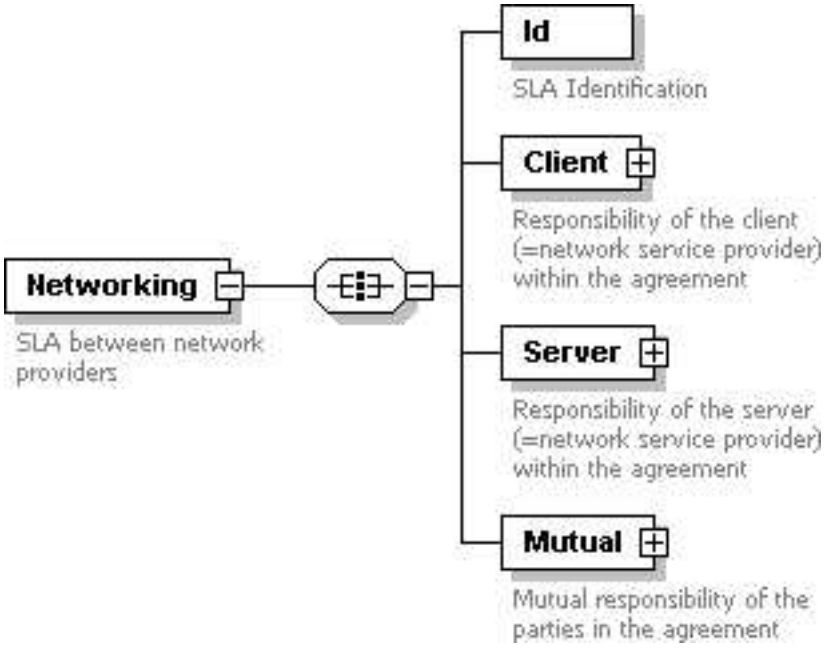



Figure 5: General structure for a *Networking* SLA

In Chapter 5, then, we present several examples of SLAs that were written in SLAng to express QoS concerns of different parties involved in our case study.

## 4.2 Networking SLA

A *Networking* SLA is contracted between network providers (Figure 5).

First of all, we identify the basic information to be handled by Service Level Specifications (SLS) [4, 22], when considering IP services to be provided together with a given quality of service (QoS). Since IP services are defined by a standard formalism for a set of basic parameters, from a technical standpoint, this information can be mapped to the elementary contents of an SLS.

According to Tequila specifications [22], an SLS is composed of the following parameters:

- SLS Identification
- Scope
- Flow Identification

- Traffic Envelope and Traffic Conformance
- Excess Treatment
- Performance Guarantees
- Service Schedule
- Reliability

Hence, we modelled the XML schema according to this structure, adding constructs for expressing end-point descriptions and contractual statements.

In the following sections, we describe elements and attributes in detail, providing the XML schema representation for each of them. Elements derived from Tequila specifications are described using their released definition([22]).

### 4.2.1 SLA Identification

The SLA Identification is a field used by the service provider and the customer to identify the SLA and the service the SLA is related to (Figure 5).

$$\text{SLA Identification} = (\text{SLA Id}, \text{Service Id}) \tag{1}$$

**SLA Id** This is the parameter identifying the SLA.

**Service Id** This the parameter identifying the service the SLA is related to.

The SLA Identification is mainly dedicated to the classification of multiple SLAs that refer to a service and/or compose it. This field is present in every type of SLA and we will not repeat its definition in the subsequent sections. What we say here is, hence, valid for each of them.

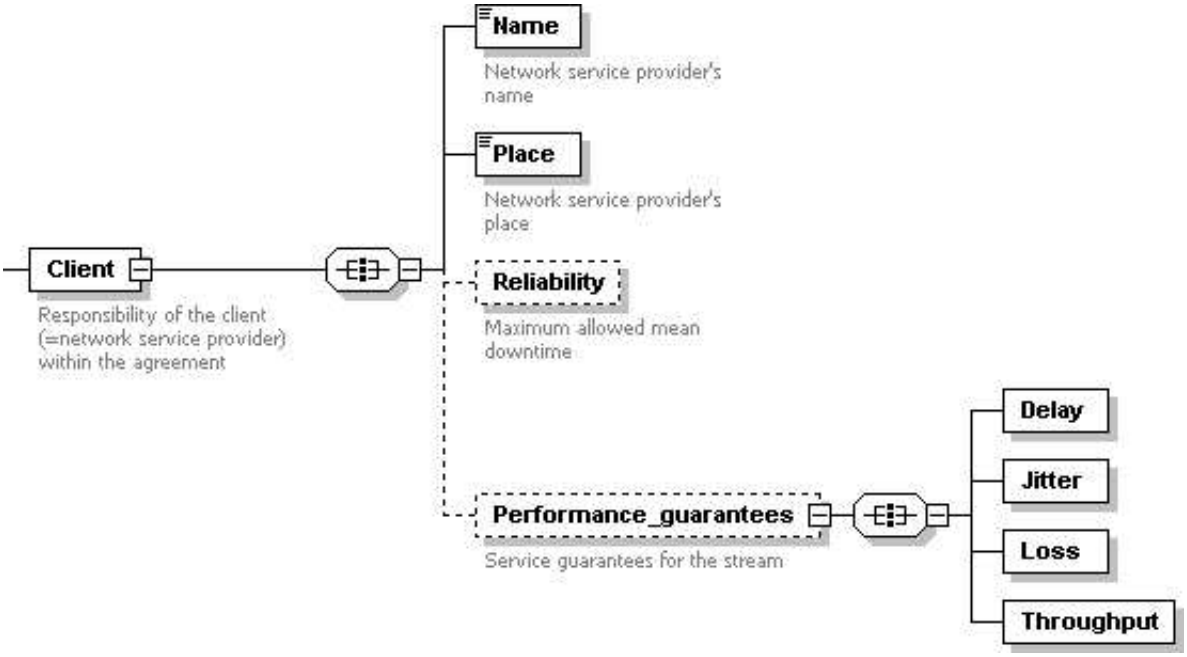



Figure 6: Client responsibilities in a *Networking* SLA.

### 4.2.2 Client responsibilities

#### Name

Name of the network service provider acting as a client in the agreement (Figure 6). It is a simple element, specified as a string.

#### Place

Geographical location of the network service provider acting as a client in the agreement (Figure 6). It is a simple element, specified as a string.

#### Reliability

Reliability indicates the maximum allowed mean downtime per year (MDT) and the maximum allowed time to repair (TTR) in case of service breakdown (e.g. in case of cable cut).

Figure 6 presents this complex element in its XML schema representation. MDT and TTR are its attributes (attributes are not shown in W3C schema design diagrams).

The Mean Down Time is expressed in minutes per year and the Maximum Time To Repair is expressed in seconds (both of them are unsignedInt).

### Performance guarantees

The performance parameters describe the service guarantees the network offers to the customer for the packet stream described by the Flow Id and within the limits of the geographical/topological extent given by the scope.

There are four performance parameters:

- one-way transit delay, optional quantile
- packet delay variation or jitter, optional quantile
- packet loss
- throughput

Delay, jitter and packet loss guarantees are for the in-profile traffic in case of binary conformance testing. For multi-level (n) conformance testing, delay, jitter and loss guarantees may be specified for each conformance level-i, except the last one (n). For example if n = 3, one can have a delay guarantee for the "conformance level-1" packets and a different delay guarantee for the "conformance level-2" packets. No guarantees are given for excess ("conformance level-n") traffic. The throughput is an overall guarantee for the IP packet stream, independent of a particular level (see below).

In our context, definitions always consider the measurable performance parameters related to the packet stream specified by the Flow Id. For simplicity, they are given for binary conformance testing (n=2), but generalisation is straightforward.

The delay and jitter indicate respectively the maximum packet transfer delay and packet transfer delay variation from ingress to egress. Delay and jitter may either be specified as worst case deterministic bounds or as quantiles. Indeed, the worst case delay/jitter bounds will be very rare events and customers may

find measurements of e.g. 99.5th percentile a more relevant empirical gauge of delay/jitter.

Suppose e.g. that the SLS specifies the couple (delay = 10ms, quantile = 10E-3). Then the probability that the transfer delay of a packet (between ingress-egress) is larger than 10ms, is less than 10E-3.

The above syntax for delay/jitter can be generalised by specifying in the SLS an array of e.g. N (delay/jitter, quantile)-couples. The more couples, the better the delay probability tail distribution can be approximated.

The packet loss probability is ratio of the lost in-profile packets between ingress and egress and the offered in-profile packets at ingress.

$$\text{packet loss} = \frac{\text{lost packets between (and including) ingress and egress}}{\text{offered (injected) packets at ingress}} \qquad (2)$$

The throughput is the rate measured at egress counting all packets identified by Flow Id. Notice that all packets, independently of their conformance level (in/out-of-profile) contribute. Indeed, if the customer wants a throughput guarantee, then he does not care whether in- or out-profile packets are dropped, but is only interested in the overall throughput of its packet stream.

The following is the relation with the Traffic Conformance Parameters (Section 4.2.4) in case of a binary-based conformance testing algorithm:

- The Traffic Conformance Algorithm (and parameters) must be specified when guaranteeing delay/jitter or packet loss, i.e. if one of these performance parameters is quantified in the SLS. Conformance testing is required because the delay/jitter and loss guarantees are only for the stream of in-profile packets.

- When only guaranteeing a throughput, or if non-of the other performance parameters is quantified, the traffic conformance algorithm may be specified. It is not required to specify the Conformance Algorithm, because the

eventual throughput guarantee does not require the strict distinction between in/out-of-profile traffic. However, the network operator will probably protect his network by implementing a Traffic Conditioner at Ingress and specifying the token policing rate (r) almost equal to the throughput guarantee R. He may or may not tag/mark excess traffic, according to his own internal policy rules.

Note on the relation between throughput R, packet loss p and excess treatment in case of a binary-based conformance testing algorithm. First consider the case where excess traffic is dropped (or shaped to in-profile) based on the token bucket (b,r) traffic conformance algorithm. As only in-profile packets are allowed at ingress, the following equality holds:

$$\text{throughput R} = (1 - p) \cdot \text{token rate r} \tag{3}$$

Thus the throughput guarantee can be derived from the loss probability and token rate and is therefore not an independent parameter.

If excess traffic is allowed (and marked accordingly), then throughput is an independent parameter because it also takes into account the out-of-profile packets (measured at egress). One has obviously the inequality:

$$\text{throughput R} \geq (1 - p) \cdot \text{token rate r} \tag{4}$$

A performance parameter is said to be quantified if its value is specified to a numeric, quantitative value. The service guarantee offered by the SLS is said to be quantitative if at least one of the 4 performance parameters is quantified. If none of the SLS performance parameters are quantified, then the performance parameters delay and packet loss may be qualified. Possible qualitative values for delay and/or loss are high, medium and low. SLAng can express both quantitative and qualitative performance guarantees.

Relative delay guarantees:

**Combination table**

| | **delay** | **low** | **medium** | **high** |
|---|---|---|---|---|
| loss | low | gold green | silver green | bronze green |
| | medium | gold yellow | silver yellow | bronze yellow |
| | high | gold red | silver red | bronze red |

*Source:* RFC 2026

Table 1: Combination table.

**gold service** value = low

**silver service** value = medium

**bronze service** value = high or not indicated

Relative loss guarantees:

**green service** value = low

**yellow service** value = medium

**red service** value = high or not indicated

The quantification of relative difference between $< high/medium/low >$ is a matter of provider's policy (e.g. high = 2 x medium ; medium = 2 x low).

The taxonomy in Table 1 yields the combinations of qualitative services.

The service guarantee offered by the SLS is said to be qualitative if it is not quantitative and either delay or loss (in a non-exclusive way) are qualified to medium or low, i.e. excluding bronze/red from the above.

The service guarantee offered by the SLS is said to be best-effort if it is not quantified nor qualified.

### 4.2.3 Server responsibility

#### Name

Name of the network service provider acting as a server in the agreement (Figure 7). It is a simple element, specified as a string.

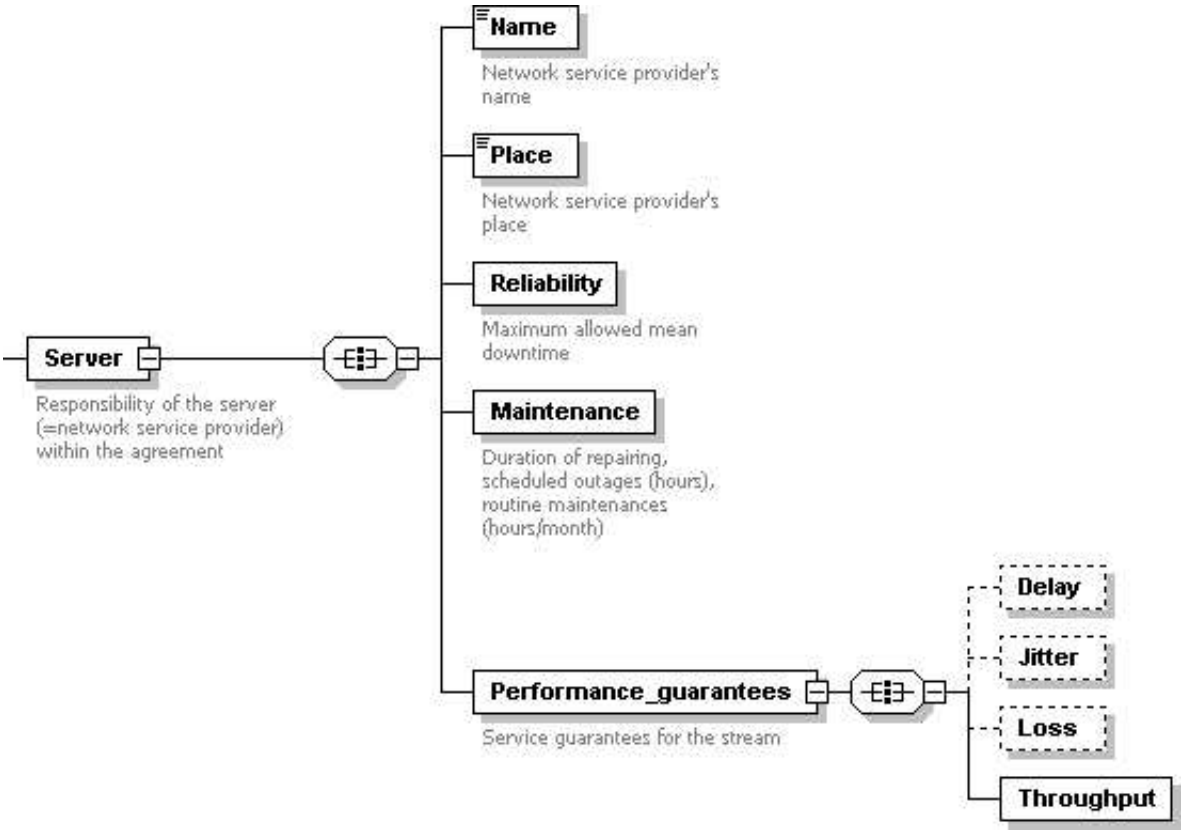



Figure 7: Server responsibilities in a *Networking* SLA.

#### Place

Geographical location of the network service provider acting as a server in the agreement (Figure 7). It is a simple element, specified as a string.

#### Reliability

This parameter has been discussed in Section 4.2.2. Refer to this Section for details on its SLAng construction.

#### Maintenance

Maintenance is described using a homonymous XML complex element, further specified by three attributes: recovery_time, scheduled_outages and routine_maintenance. Mean time to recover from failure is a required parameter. Possible planned outages and routine maintenances can be optionally specified.

#### Performance guarantees

This parameter has been discussed in Section 4.2.2. Refer to this Section for details on its SLAng construction.

### 4.2.4 Mutual responsibility

#### Service schedule

The service schedule indicates the start and the end of the service, i.e. when the service is available.

The Service schedule may be specified with the following parameters:

$$\text{Service schedule} = (\text{Start date, End date}) \tag{5}$$

**Start date** Date and hour from which the service becomes available.

**End date** Date and hour from which the service becomes unavailable.

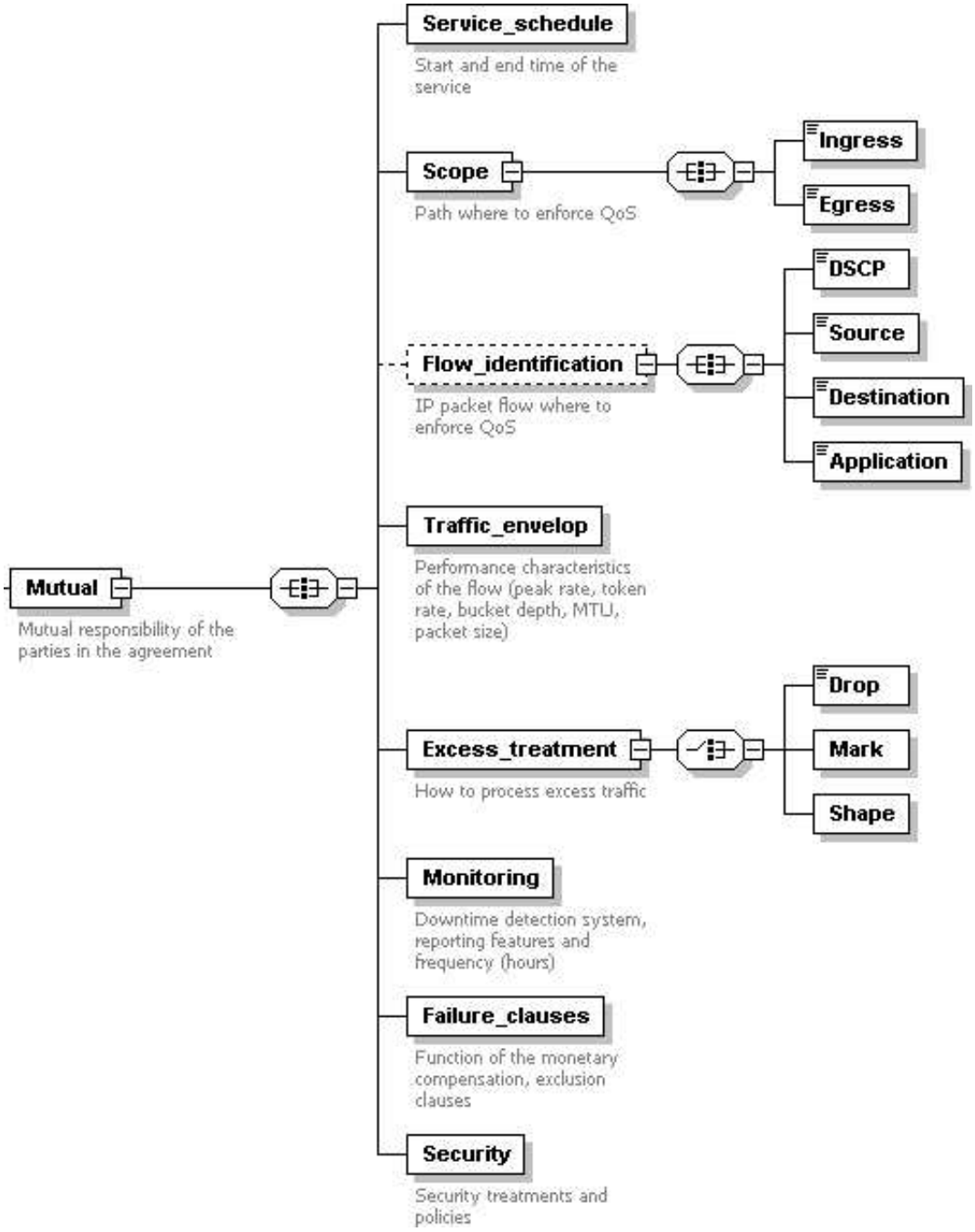



Figure 8: Mutual responsibilities in a *Networking* SLA.

Start date and End date must be specified and End date must be greater than End date. Service schedule "from now on", i.e. *[now, infinity]*, can be captured by putting the above to their full range.

An SLA is active between the Start date and the End date.

### Scope

The scope of an SLA associated to a given service offering indicates where the Quality of Service (QoS) policy for that specific service offering is to be enforced. Therefore the scope uniquely identifies the geographical/topological region over which the QoS is to be enforced by indicating the boundaries of that region.

The associated scope of the SLS must be expressed by a couple of ingress and egress interfaces. Ingress/egress denote respectively the entry/exit points of the IP packets relative to the network domain.

$$Scope = (ingress, egress), \tag{6}$$

with ingress/egress defined as:

**Ingress** interface identifier |set of interface identifiers |any

**Egress** interface identifier |set of interface identifiers |any

From now on the following remarks apply:

- "|" denotes an exclusive OR.
- "any" is logically equivalent with unspecified.

The following combinations of (ingress, egress) interfaces are allowed:

**(1,1)** one-to-one communication (Pipe)

**(1,N)** one-to-many communication ($N > 1$) (Hose)

**(1,any)** one-to-any communication

**(N,1)** many-to-one communication ($N > 1$) (Funnel)

**(any,1)** any-to-one communication

The above taxonomy excludes the many-to-many communication (M,N). Either ingress or egress must be specified to exactly one interface identifier (with a non-exclusive OR). Many-to-many communication (M,N) can be decomposed into M times one-to-many communication (1,N).

This taxonomy avoids all ambiguity about the IP flow (defined as a set of IP datagrams sharing at least one common characteristic, like e.g. the same [source address; destination address] pair), and its corresponding identification (see Flow identification and Traffic envelope). If the ingress is a single interface identifier, then the traffic envelop and flow id concerns the incoming IP packet stream at the unique ingress point. If only the egress is a single interface, i.e. (N |any,1), then the traffic envelop and flow id concerns the outgoing (aggregate) traffic on the egress link.

In this document SLSs with an associated scope (topology) of (1,1) ; (1,N) ; (N,1) is called respectively Pipe, Hose and Funnel SLSs.

Figure 8 depicts the scope XML schema element.

An ingress (or egress) interface identifier should uniquely determine the boundary link as defined in [4] on which packets arrive/depart at the border of a DS domain. This link identifier may be an IP address, but it may also be any other mutually agreed upon identifier which uniquely identifies a boundary link. For example a layer-two identifier in case of Ethernet or unnumbered PPP-based access links in Point-to-Point Protocol.

### Flow Identification

The flow identification of an SLS associated to a given service offering indicates for which IP packets the QoS policy for that specific service offering is to be enforced.

A flow identification identifies a stream of IP datagrams sharing at least one common characteristic. An SLS contains one (and only one) flow identification,

which may formally be specified by providing one or more of the following attributes[1]

$$\text{Flow Id} = (\text{DiffServ info}, \text{source info}, \text{dest info}, \text{app info}) \quad (7)$$

**Differentiated Services information** DSCP value |set of DSCP values |any

**Source information** source address |set of source addresses |source prefixe |set of source prefixes |any

**Destination information** destination address |set of destination addresses |destination prefixe |set of destination prefixes |any

**Application information** protocol number |protocol number and source port, destination port |any

Thus, the Flow Id may be expressed by information attributes related to the source/destination nodes, the application or the DS field in the IP header. Note that the element is optional (Figure 8). In fact, only DiffServ-enabled network domains need to specify it.

The Flow Id provides the necessary information for classifying the packets at a DS boundary node. This datagram classification can either reflect a Behaviour Aggregate (BA) or Multi-Field (MF)classification. In case of MF-classification all attributes may be specified, including the DSCP field. MF classification may depict as well micro-flows as aggregate macro-flows, based on e.g. source network prefix [26].

Also the set-of semantics allows for the specification of aggregate flows. If a Flow Id is e.g. specified by a set of two IP source addresses, then any packet with either of the two concerned source addresses in its header belongs to the IP packet stream identified by Flow Id.

Finally note also that the IP routing scheme may put restrictions on combining scope and flow identification within an SLS.

[1]The Differentiated Services Code Point (DSCP) IP header field is defined in: [26]

In Figure 8 the Flow Identification XML schema element is represented.

In general, if only Flow ID is specified by source and destination IP address (IP-src, IP-dest), and the scope is unspecified, then there is no a-priori assumption about the actual ingress/egress points that this traffic will cross. Indeed, it is the responsibility of the network provider to define the most appropriate route for this traffic, by enforcing the corresponding traffic engineering and routing policies. Thus, the (ingress, egress) information (which in this case is not part of the SLS template instance) is then derived from the Flow Id and the routing policy of the network provider.

On the other hand, if both Flow Id and scope are specfied in the SLS, respectively by the pairs (IP-src, IP-dest) and (IP-ingr, IP-egr) pairs, then it is clear that the IP packets must follow the route (IP-src,...,IP-ingr,...,IP-egr,...,IP-dest). Thus the restriction is that the scope (IP-ingr, IP-egr) is part of the route from IP-scr to IP-dest.

Also it is important to remark that the exclusion of the many-to-many communication scope model puts similar constraints on the source/destination fields of the Flow Identification.

### Traffic Envelop and Traffic Conformance

The traffic envelop describes the traffic (conformance) characteristics of the IP packet stream identified by the Flow Id. The traffic envelop is a set of Traffic Conformance Parameters, describing how the packet stream should look like to get the guarantees indicated by the performance parameters (see Performance guarantees).

The Traffic Conformance Parameters are the basic input for the Traffic Conformance Algorithm. Traffic Conformance Testing is the combination of the Traffic Conformance Parameters and the Traffic Conformance Algorithm. This will be done at a DS-boundary node.

The algorithm and the conformance test can be binary-based or multi-level based. Binary Traffic Conformance Testing is a set of actions which uniquely

identifies the in-profile and out-of profile (or excess) packets of an IP stream (identified by Flow-Id). In this case, the Traffic Conformance Parameters describe the reference values the traffic (identified by the Flow ID) will have to comply with, thus yielding the notions of in and out of profile traffics. The Traffic Conformance Algorithm is the mechanism enabling unambiguously to identify all in or out of profile packets based on these Conformance parameters.

In case of multi-level (n) Traffic Conformance Testing a packet will be tagged (by the algorithm) as belonging to a particular level (1...n). Packets tagged as level n are called excess packets.

The following gives a (non-exhaustive) list of potential conformance parameters:

- Peak rate p (bits per second)
- Token bucket rate r (bits per second)
- Bucket depth b (bytes)
- Maximum Transfer Unit (MTU) M (bytes)
- Minimum packet size (bytes)

And in Figure 8 the corrisponding XML schema definition is provided.

Binary-based Traffic Conformance Testing examples:

- Conformance parameters = token bucket parameters (b,r); conformance algorithm = token bucket algorithm.
- Conformance parameters = token bucket parameters and peak rate (b,r,p) with p larger than r; conformance algorithm = the combined token bucket (b,r) and (b,p).

  The scheme permits bursty traffic to be sent, limited to a burst of b bytes, with a (long-term) average rate of r and a peak rate of no more than p.

- Conformance parameters = MTU; conformance algorithm = all packets allowed with size smaller than MTU; packets larger than MTU are fragmented or dropped.

Three-level based Traffic Conformance Testing example:

- The Two-rate Three-colour marker is based on two token buckets with rates r1 and r2 (r2 being greater than r1), containing respectively green and yellow tokens. The simplest operational mode is the colour-blank mode. A packet is tagged "green" if there are green and yellow tokens available, yellow if only yellow tokens are available and otherwise it is tagged red.

**Excess Treatment**

This section describes how the network provider will process excess traffic, i.e. out-of-profile traffic (in case of binary conformance testing) or n-level traffic (in case of n-level conformance testing). The process takes place after Traffic Conformance Testing, described previously.

Excess traffic may be dropped, shaped and/or remarked. The SLS must specify the appropriate action by the following attribute.

If Excess Treatment is not indicated, then excess traffic is dropped. Depending on the appropriate action, more parameters may be required. The following is an indication in case of binary conformance testing. Multi-level conformance testing (like the definition of a hierarchical drop preference model) may also be enforced.

- If excess traffic is dropped, then all packets marked as "out-of-profile" by the Traffic Conformance Algorithm are dropped. No extra parameters are needed.

- If excess traffic is shaped, then all packets marked as "out- of-profile" by the Traffic Conformance Algorithm are delayed until they are "in-profile". The shaping rate is the policing/token bucket rate r. The extra parameter is the buffer size of the shaper.

- If excess traffic is marked or remarked, then all packets marked as "out-of-profile" by the the Traffic Conformance Algorithm are (re-) marked with a particular DSCP-value (yellow or red). The extra parameter is the DSCP.

In Figure 8, the schema implementation is illustrated.

### Monitoring

Monitoring performance is essential to undestand if contracted service levels are respected. Customer and provider needs to agree on downtime detection systems and also on the features and the frequency of reporting. This complex element is specified by defining a set of attributes. First of all the tracking system can (optionally) be specified. It is required, then, to state method and frequency of reports and if reporting on demand is available. Optionally, security violations can be further checked.

### Failure clauses

In the event of service levels being disregarded, a compensation must be provided to the interested party. Through this element, a function of the monetary compensation can be expressed, along with exclusion clauses for the infringement of service use conditions.

A possible proposed scheme is the following. C = Credit amount based on monthly payment

$$\mathrm{C} = \mathrm{f}(\mathrm{NV}, \mathrm{F}, \mathrm{P}), \tag{8}$$

$f$ being a discrete function where NV = (100% - AV), AV = availability, P = period of day, month, year (in order to take into account peak periods) and F = frequence of NV during P.

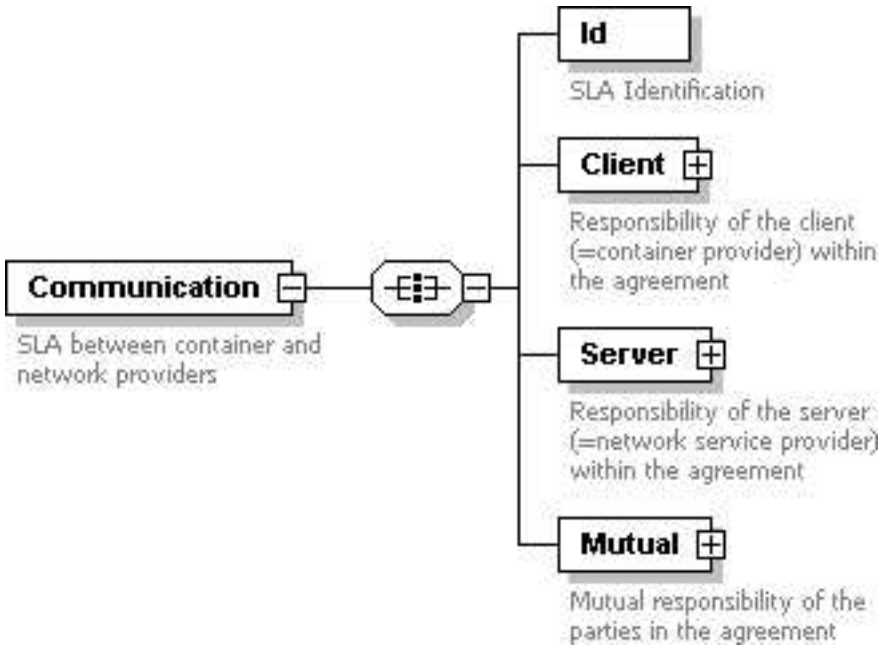



Figure 9: General structure for a *Communication* SLA.

**Security**

Security treatments and policies can be specified using this element. It is possible to state whether a firewall is used or not, together with the specification of the firewall system, whether user authentication, intrusion detection and eavesdrop prevention are enabled and if *ipsec* is provided as an option. In this context, all of these attributes are required.

## 4.3 Communication SLA

A *Communication* SLA specifies contracts between a container and a network provider (Figure 9). Such an agreement is contracted by Applcation Service Providers (ASPs) that need network services from Internet Service Providers (IPSs).

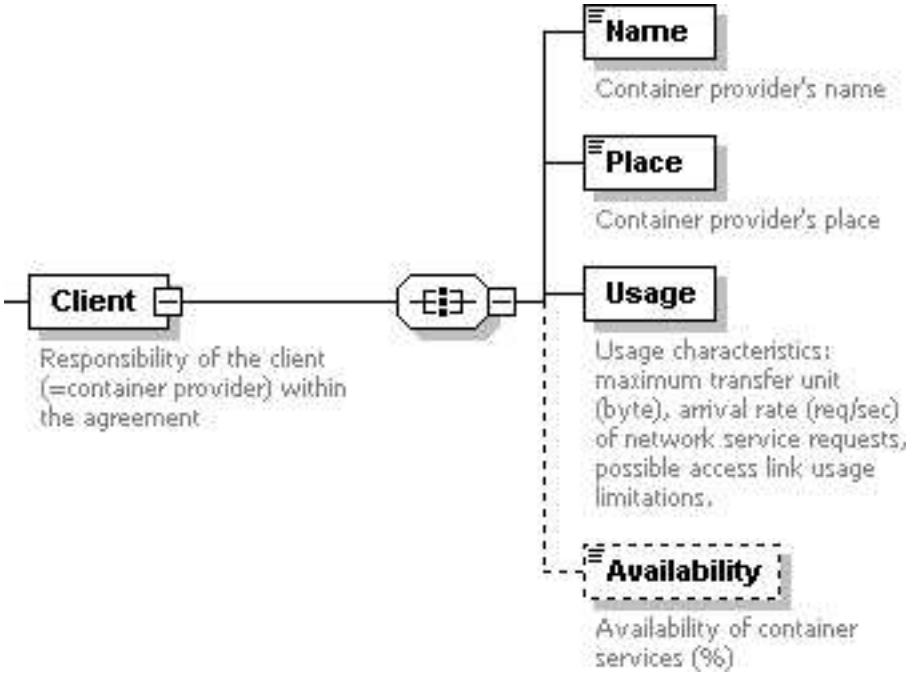



Figure 10: Client responsibilities in a *Communication* SLA.

### 4.3.1 Client responsibilities

#### Name

Name of the container provider acting as a client in the agreement (Figure 10). It is a simple element, specified as a string.

#### Place

Geographical location of the container provider acting as a client in the agreement (Figure 10). It is a simple element, specified as a string.

#### Usage

Usage characteristic can be specified through this element. Maximum transfer unit in bytes and arrival rate in number of requests per second are required attributes. Optionally, access link limitation can be defined as a percentage of the full utilisation.

#### Availability

Optional, simple element expressing the percentage of container availability over one year.

### 4.3.2 Server responsibility

#### Name

Name of the network service provider acting as a server in the agreement (Figure 11). It is a simple element, specified as a string.

#### Place

Geographical location of the network service provider acting as a server in the agreement (Figure 11). It is a simple element, specified as a string.

#### Maintenance

Maintenance is described using a homonymous XML complex element, further specified by three attributes: recovery_time, scheduled_outages and routine_maintenance. Mean time to recover from failure is a required parameter. Possible planned outages and routine maintenances can be optionally specified.

#### Reliability

Maximum allowed cumulated downtime per year in hours can be expressed through this required element.

#### Performance guarantees

Specification of delay and jitter requirements in milliseconds, packet loss as a percentage and bandwidth in Megabytes per second is possible, only the last one being required. Refer to Section 4.2.2 for more details about performance guarantees specifications.

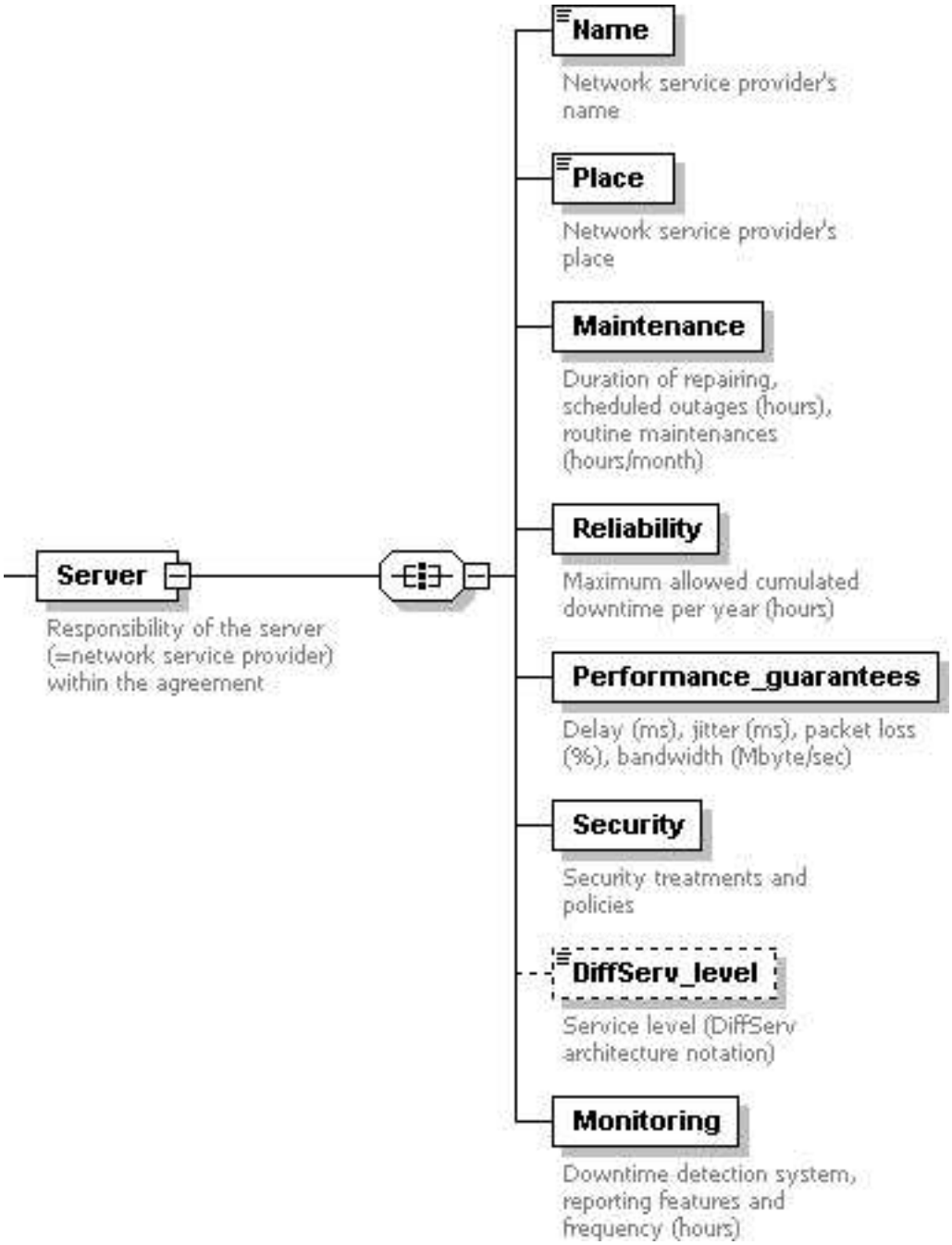



Figure 11: Server responsibilities in a *Communication* SLA.

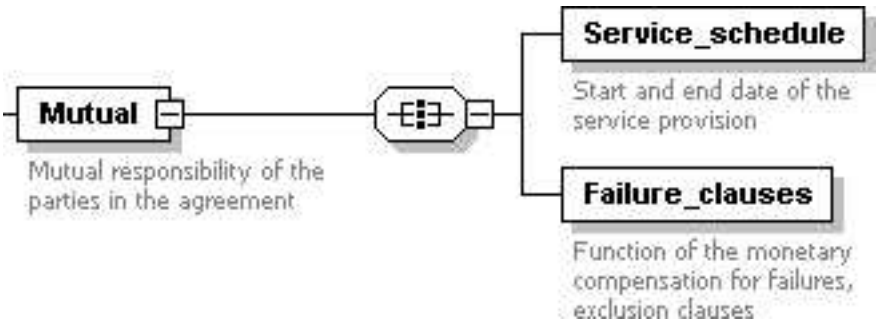



Figure 12: Mutual responsibilities in a *Communication* SLA.

### Security

Security treatments and policies can be specified using this element. It is possible to state whether a firewall is used or not, together with the specification of the firewall system, whether user authentication, intrusion detection and eavesdrop prevention are enabled and if *ipsec* is provided as an option. In this context, all of these attributes are required.

### DiffServ level

Service level according to Differenciated Services architecture notation can be optionally contracted with the client, provided that this facility is enabled by the network provider.

### Monitoring

Customer and provider needs to agree on downtime detection systems and also on the features and the frequency of reporting. This complex element is specified by defining a set of attributes. First of all the tracking system can (optionally) be specified. It is required, then, to state method and frequency of reports and if reporting on demand is available. Optionally, security violations can be further checked.

### 4.3.3 Mutual responsibility

#### Service schedule

The service schedule indicates the start and the end of the service, i.e. when the service is available.

The Service schedule may be specified with the following parameters:

$$\text{Service schedule} = (\text{Start date, End date}) \tag{9}$$

**Start date** Date and hour from which the service becomes available.

**End date** Date and hour from which the service becomes unavailable.

Start date and End date must be specified and End date must be greater than End date. Service schedule "from now on", i.e. *[now, infinity]*, can be captured by putting the above to their full range.

An SLA is active between the Start date and the End date.

#### Failure clauses

In the event of service levels being disregarded, a compensation must be provided to the interested party. Through this element, a function of the monetary compensation can be expressed, along with exclusion clauses for the infringement of service use conditions.

A possible proposed scheme is the one presented in Section 4.2.4.

## 4.4 Container SLA

A *Container* SLA specifies a contract between container providers (Figure 13). Two Applcation Service Providers want to enter such an agreement to put component replication in place. Software components can be replicated at different locations to enable high availability and load balancing.

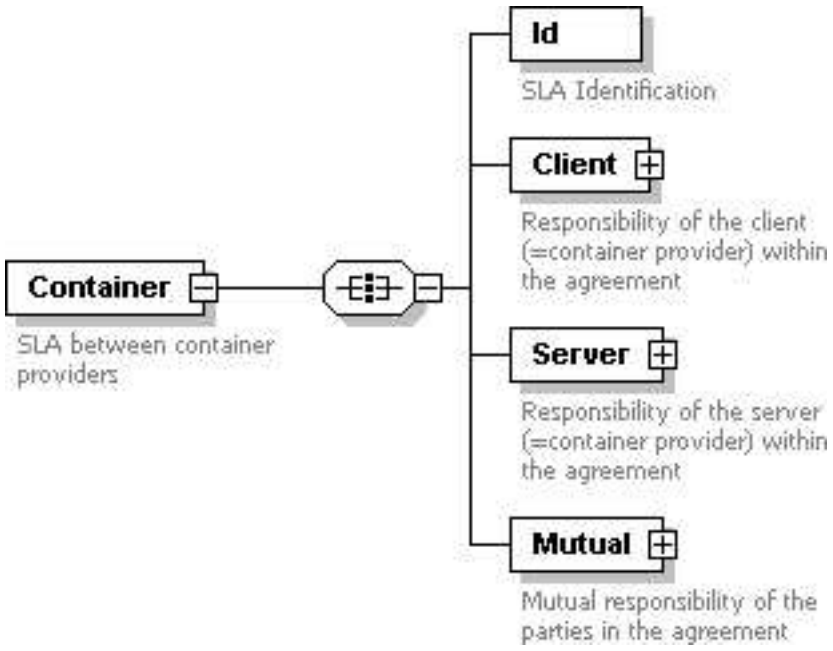



Figure 13: General structure for a *Container* SLA

### 4.4.1 Client responsibilities

#### Name

Name of the container provider acting as a client in the agreement (Figure 14). It is a simple element, specified as a string.

#### Place

Geographical location of the container provider acting as a client in the agreement (Figure 14). It is a simple element, specified as a string.

#### Availability

Simple element expressing the percentage of container availability over one year.

#### Disk space

Disk space reserved by the container provider acting as a client within the agreement. This element is optional.

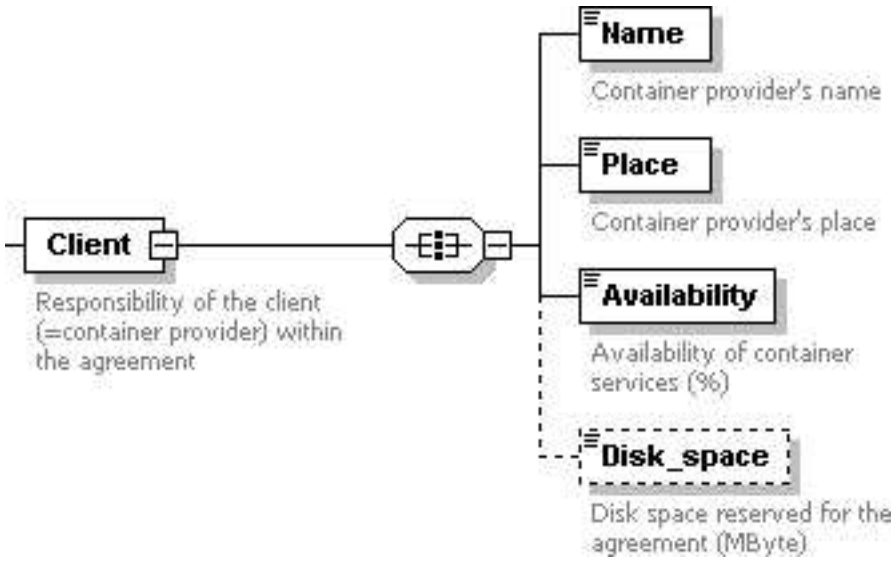



Figure 14: Client responsibilities in a *Container* SLA.

### 4.4.2 Server responsibility

#### Name

Name of the container provider acting as a server in the agreement (Figure 15). It is a simple element, specified as a string.

#### Place

Geographical location of the container provider acting as a server in the agreement (Figure 15). It is a simple element, specified as a string.

#### Availability

Simple element expressing the percentage of container availability over one year.

#### Disk space

Disk space reserved by the container provider acting as a server within the agreement.

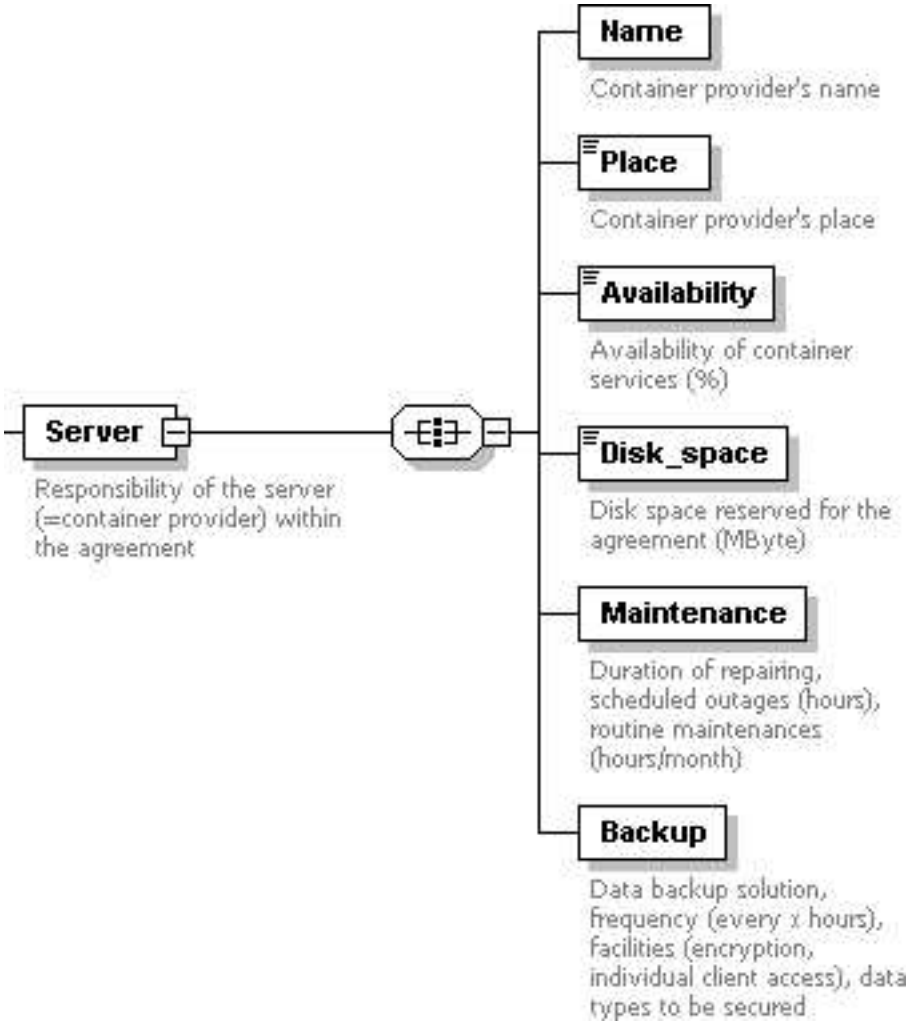



Figure 15: Server responsibilities in a *Communication* SLA.

#### Maintenance

Maintenance is described using a homonymous XML complex element, further specified by three attributes: recovery_time, scheduled_outages and routine_maintenance. Mean time to recover from failure is a required parameter. Possible planned outages and routine maintenances can be optionally specified.

#### Backup

Data backup solutions and facilities can be described through this complex element. It includes attributes to specify backup system, frequency (both complete and incremental backup intervals), data types to be backed-up and archiving form. Client access (possibly individual) and backup encryption can be offered by using the related attributes. Only *solution*, *complete_backup_interval* and *data_type* attributes are required in this context.

### 4.4.3 Mutual responsibility

For the definition of Service schedule, Security, Monitoring and Failure clauses refer to Section 4.3.3, 4.3.2, 4.3.2, 4.2.4 respectively, as they are exactly the same in this context. The remaining elements in Figure 16 are presented in the following three sections.

#### Transactions

Responsibilities on transaction quality can be stated using the attributes of this complex element. They include throughput in Megabytes per second, response time in milliseconds, abort and succesful transaction ratio (as a percentage).

#### Clients

Through this simple element, one can specify the number of supported clients for the provision concerned with the agreement.

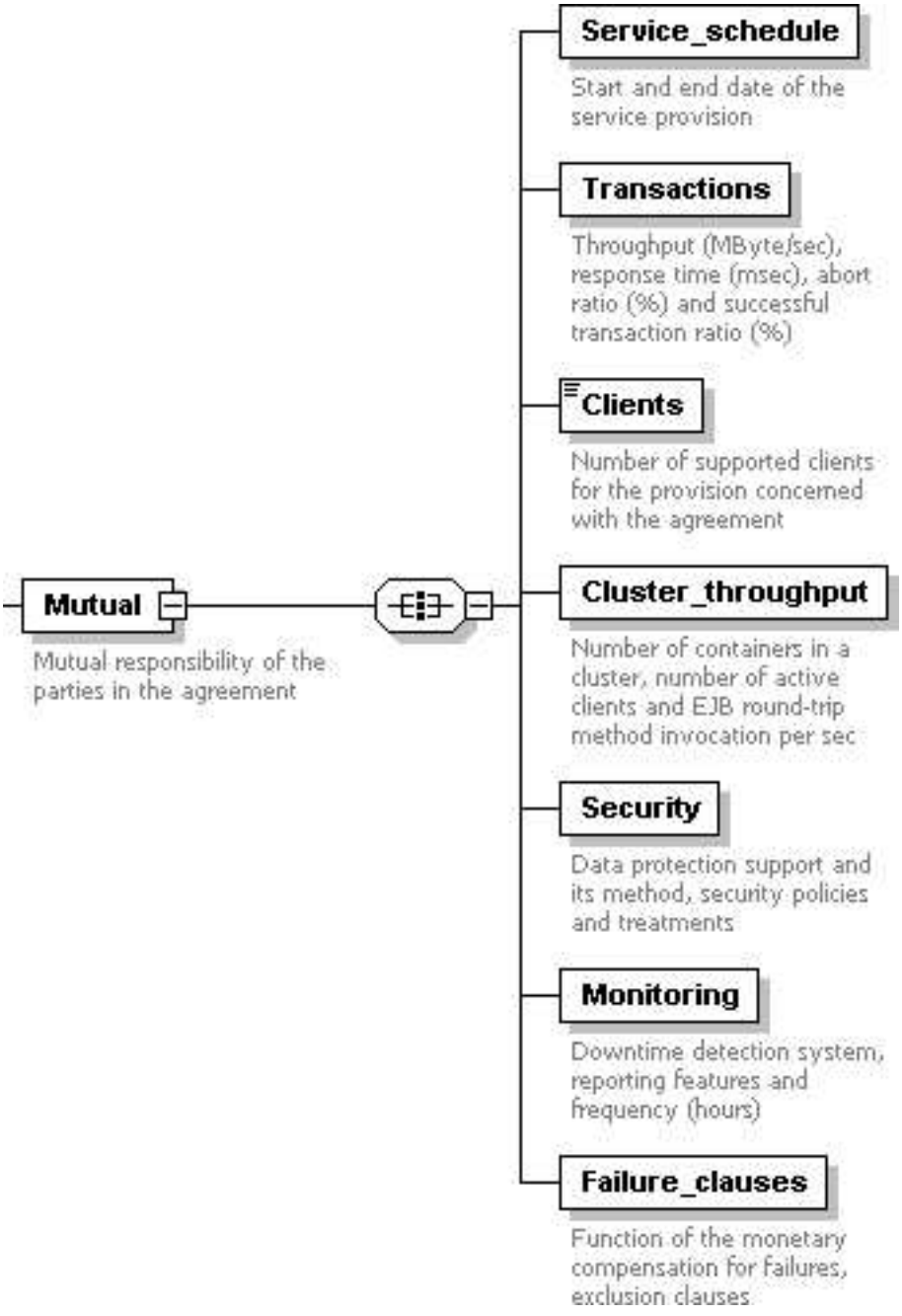



Figure 16: Mutual responsibilities in a *Container* SLA.

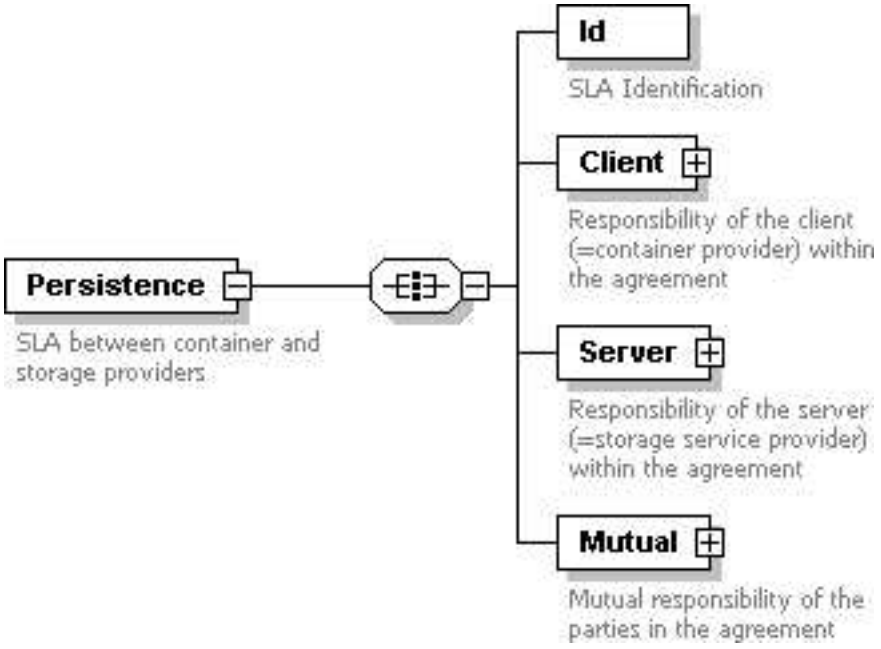



Figure 17: General structure for a *Persistence* SLA

**Cluster throughput**

Three attributes define this element: number of containers in a cluster, number of active clients and EJB round-trip method invocation per second.

## 4.5 Persistence SLA

A *Persistence* SLA is contracted between a container and a storage provider (Figure 17).

### 4.5.1 Client responsibilities

**Name**

Name of the container provider acting as a client in the agreement (Figure 18). It is a simple element, specified as a string.

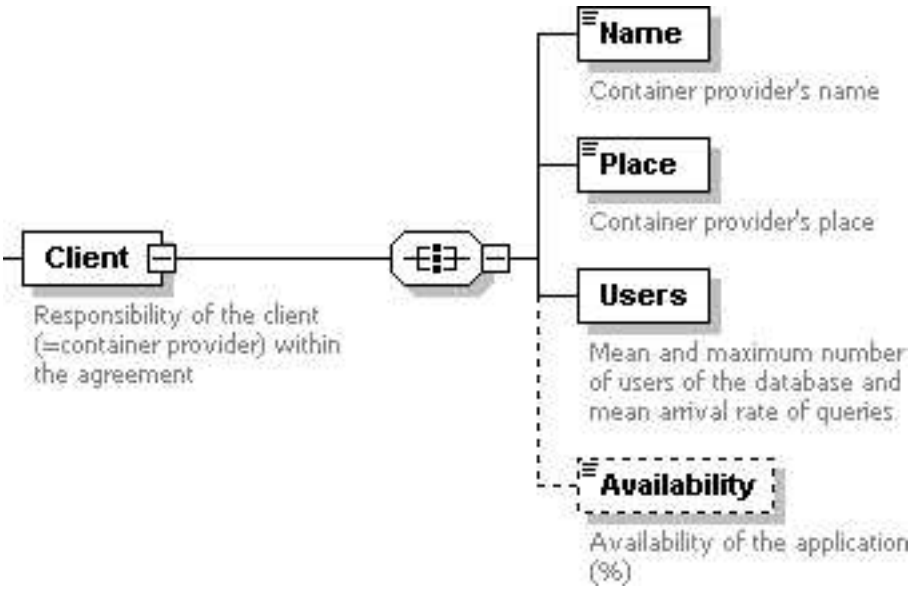



Figure 18: Client responsibilities in a *Persistence* SLA.

### Place

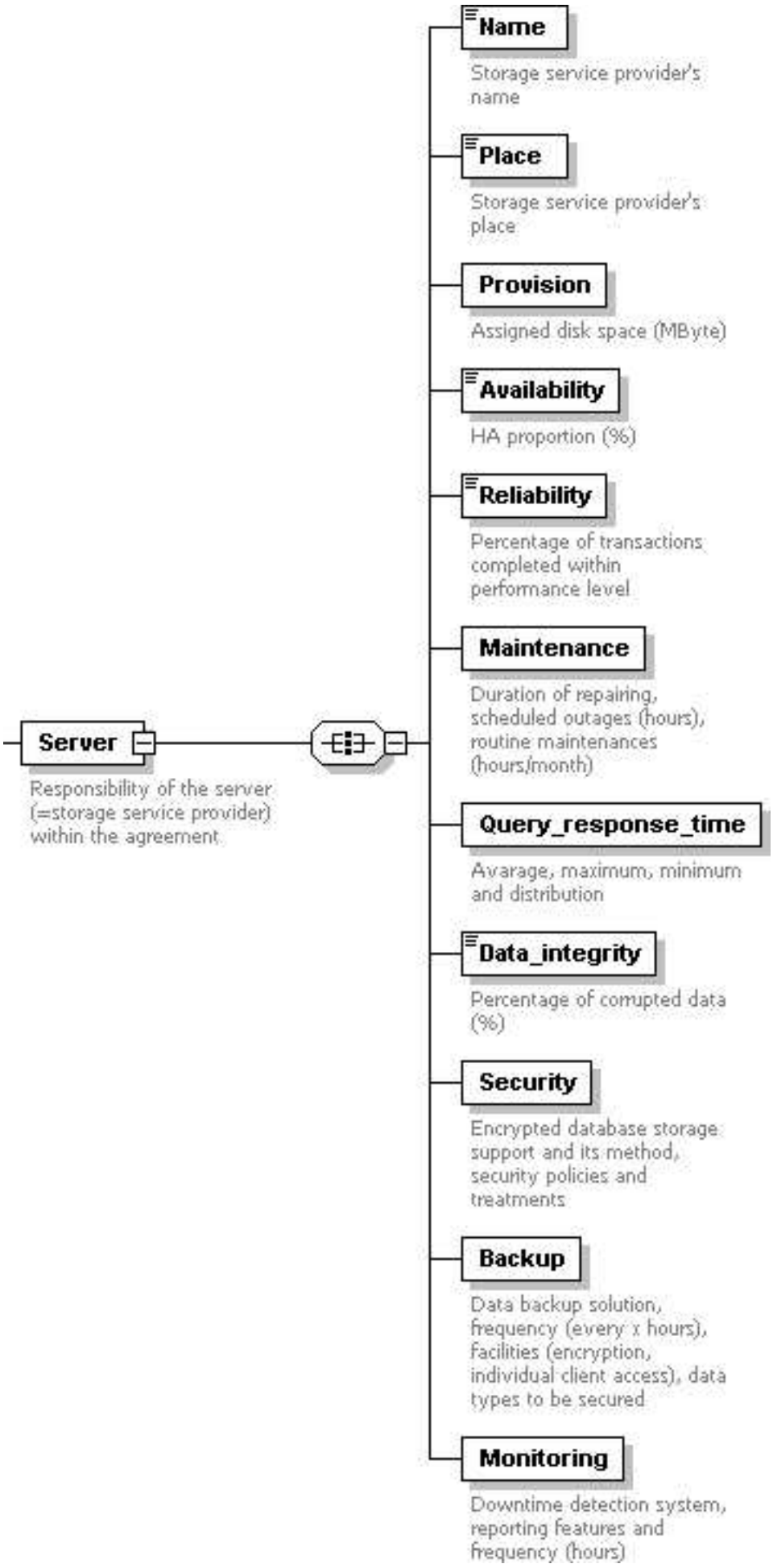
Geographical location of the container provider acting as a client in the agreement (Figure 18). It is a simple element, specified as a string.

### Users

Complex element describing the extimated number of users accessing the storage service. Mean and maximum number can be defined together with the arrival rate of queries.

### Availability

Simple optional element expressing the percentage of container availability over one year.


Server
Responsibility of the server (=storage service provider) within the agreement
Name
Storage service provider's name
Place
Storage service provider's place
Provision
Assigned disk space (MByte)
Availability
HA proportion (%)
Reliability
Percentage of transactions completed within performance level
Maintenance
Duration of repairing, scheduled outages (hours), routine maintenances (hours/month)
Query_response_time
Avarage, maximum, minimum and distribution
Data_integrity
Percentage of corupted data (%)
Security
Encrypted database storage support and its method, security policies and treatments
Backup
Data backup solution, frequency (every x hours), facilities (encryption, individual client access), data types to be secured
Monitoring
Downtime detection system, reporting features and frequency (hours)

### 4.5.2 Server responsibility

#### Name

Name of the storage service provider acting as a server in the agreement (Figure 19). It is a simple element, specified as a string.

#### Place

Geographical location of the storage service provider acting as a server in the agreement (Figure 19). It is a simple element, specified as a string.

#### Provision

Assigned disk space for the storage service in Megabytes can be expressed through this element.

#### Availability

See previous Section.

#### Reliability

This simple element defines the number of transactions completed within performance level as a percentage.

#### Maintenance

Maintenance is described using a homonymous XML complex element, further specified by three attributes: recovery_time, scheduled_outages and routine_maintenance. Mean time to recover from failure is a required parameter. Possible planned outages and routine maintenances can be optionally specified.

#### Query response time

This is a crucial index for database performance. It largely depends on the query. Contractors can negotiate to reference particolar queries and state that specifing

an attribute of this complex element. Average query response time is a required attribute. Optionally, maximum, minimum and its distribution can be expressed, for a particularly precise quality guarantee.

#### Data integrity

This simple element express the percentage of non-corrupted data.

#### Security

Refer to a previously given description. All the attributes of this element are required in this context.

#### Backup

Refer to a previously given description. Only *solution*, *complete_backup_interval* and *data_type* attributes are required in this context.

#### Monitoring

Refer to a previously given description. *Report_method*, *report_frequency* and *reporting_on_demand* attributes are required in this context.

### 4.5.3 Mutual responsibility

#### Service schedule

Refer to a previously given description.

#### CPU utilisation

Simple element describing mean CPU utilisation as a percentage.

#### Memory usage

Simple element describing mean memory usage as a percentage.

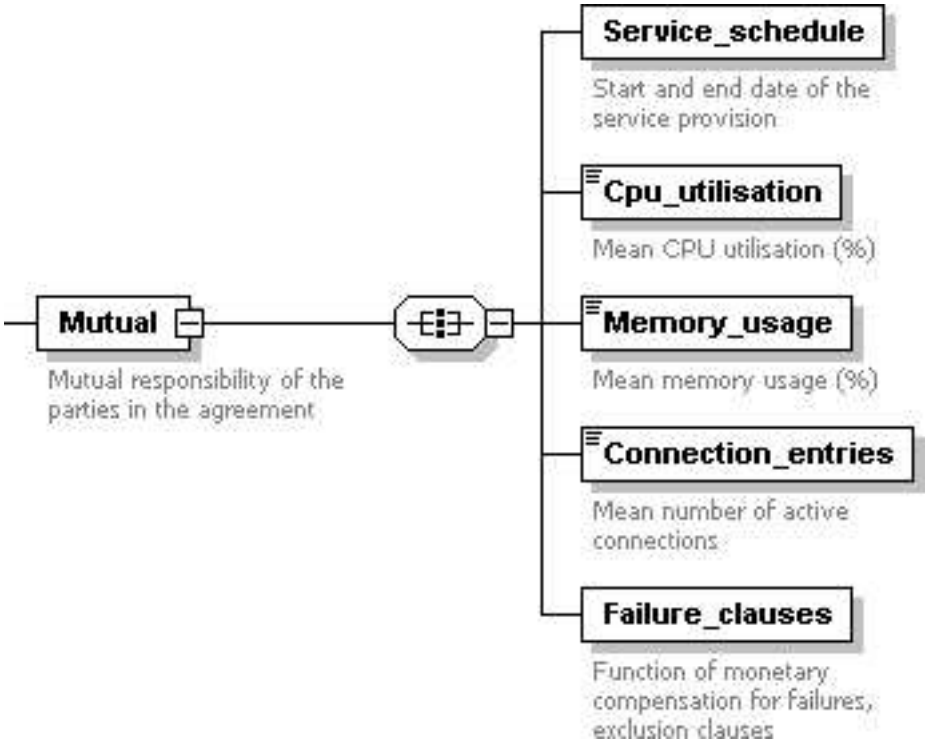



Figure 20: Mutual responsibilities in a *Persistence* SLA.

#### Connection entries

Mean number of active connections can be expressed through this element.

#### Failure clauses

Refer to a previously given description.

## 4.6 Hosting SLA

A *Hosting* SLA is contracted between a container and a component providers (Figure 21).

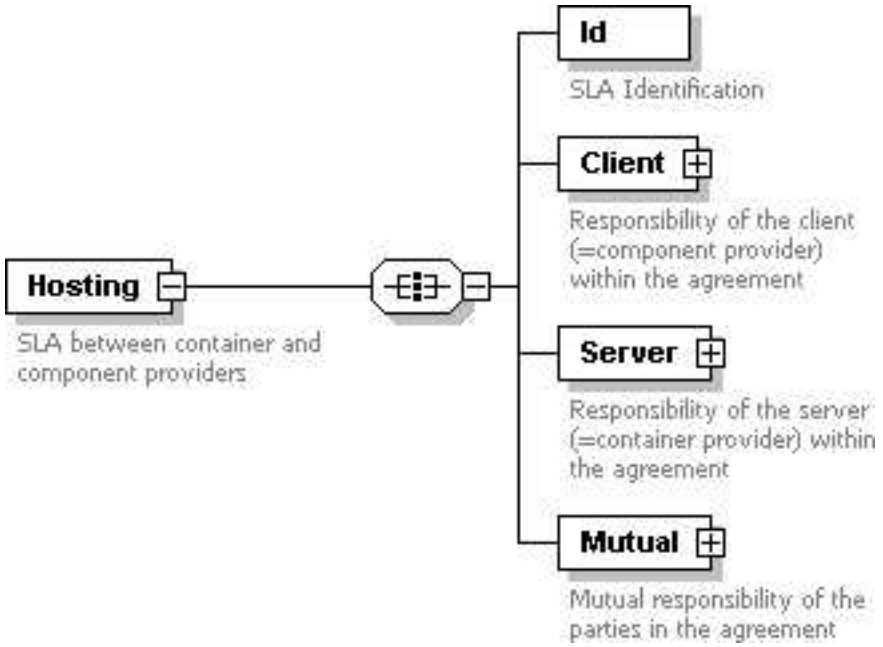



Figure 21: General structure for a *Hosting* SLA

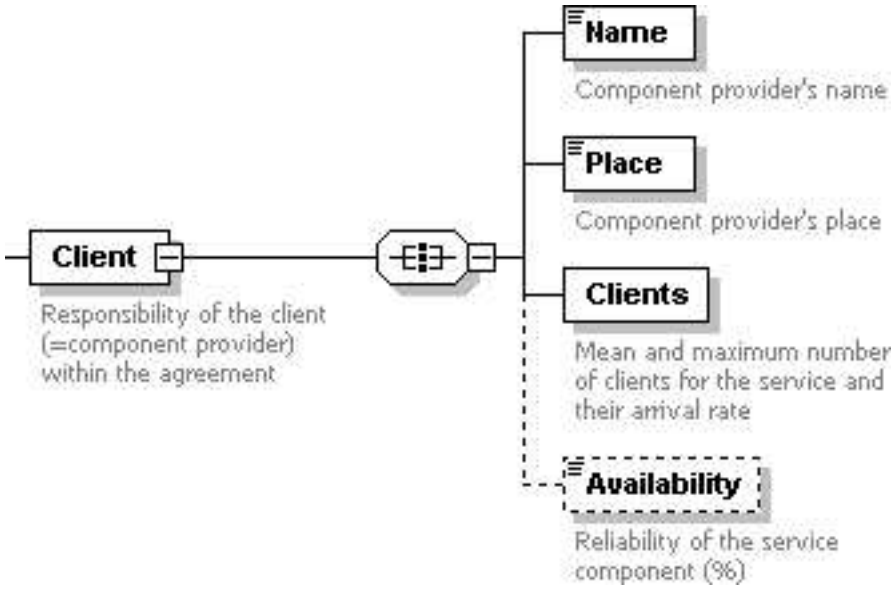



Figure 22: Client responsibilities in a *Hosting* SLA.

### 4.6.1 Client responsibilities

#### Name

Name of the component provider acting as a client in the agreement (Figure 22). It is a simple element, specified as a string.

#### Place

Geographical location of the component provider acting as a client in the agreement (Figure 22). It is a simple element, specified as a string.

#### Clients

Complex element describing the extimated number of clients accessing container services. Mean and maximum number can be defined together with the arrival rate of queries.

#### Availability

Simple optional element expressing the percentage of component service availability over one year.

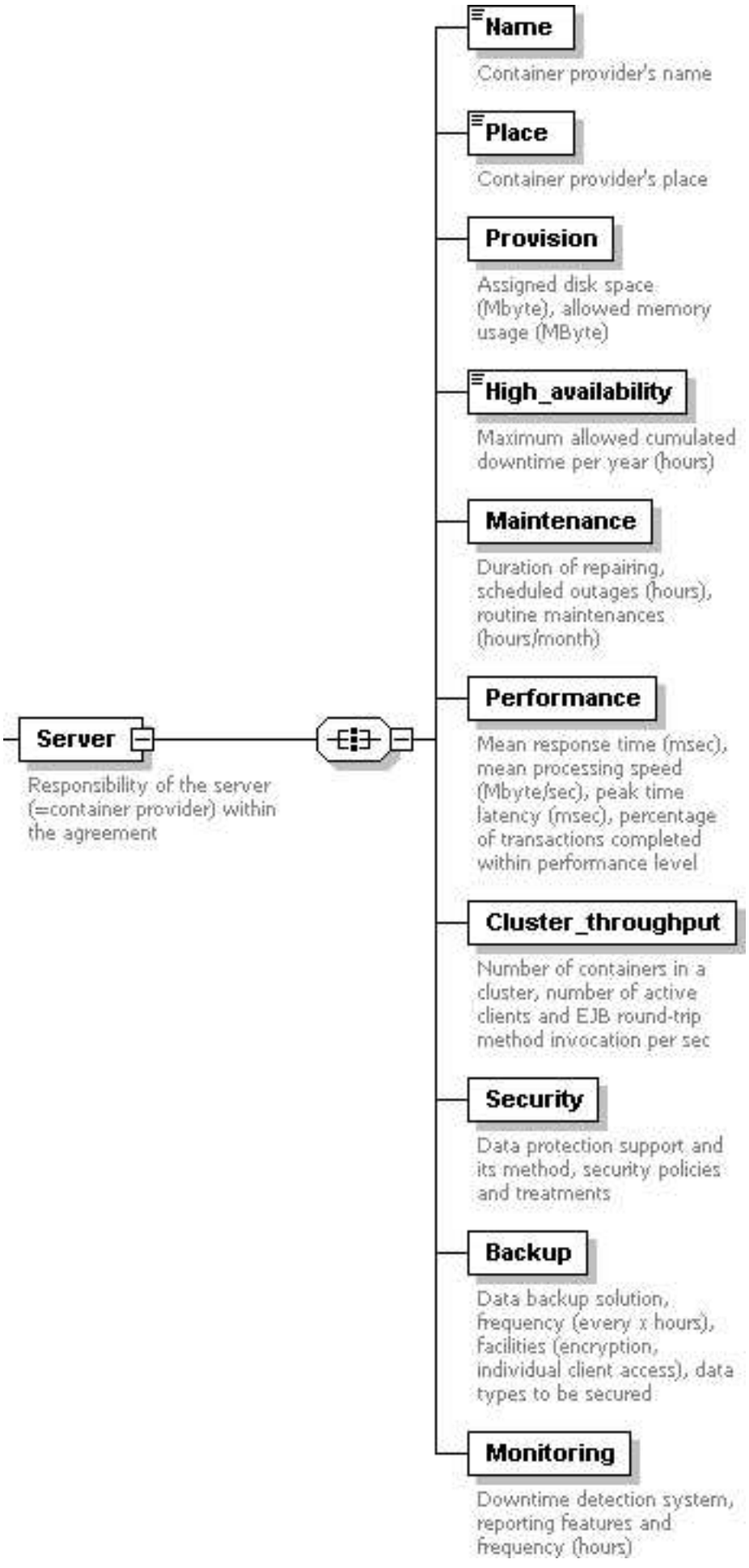
### 4.6.2 Server responsibility

#### Name

Name of the container provider acting as a server in the agreement (Figure 15). It is a simple element, specified as a string.

#### Place

Geographical location of the container provider acting as a server in the agreement (Figure 15). It is a simple element, specified as a string.


Server
Responsibility of the server (=container provider) within the agreement
Name
Container provider's name
Place
Container provider's place
Provision
Assigned disk space (Mbyte), allowed memory usage (MByte)
High_availability
Maximum allowed cumulated downtime per year (hours)
Maintenance
Duration of repairing, scheduled outages (hours), routine maintenances (hours/month)
Performance
Mean response time (msec), mean processing speed (Mbyte/sec), peak time latency (msec), percentage of transactions completed within performance level
Cluster_throughput
Number of containers in a cluster, number of active clients and EJB round-trip method invocation per sec
Security
Data protection support and its method, security policies and treatments
Backup
Data backup solution, frequency (every x hours), facilities (encryption, individual client access), data types to be secured
Monitoring
Downtime detection system, reporting features and frequency (hours)

### Provision

Assigned disk space and allowed memory usage for container services (both in Megabytes) can be expressed through this element.

### High availability

By definition, maximum allowed downtime per year in hours [17].

### Maintenance

Refer to a previously given description. Mean time to recover from failure is the only required parameter in this context.

### Performance

In this context, performace is defined by mean response time in milliseconds, mean processing speed in Megabytes per second, peak time latency in milliseconds, percentage of transactions completed within performance level.

### Cluster throughput

Cluster throughput is defined by three parameters, which are the homonymous attributes of this complex element: number of containers in a cluster, number of active clients and EJB round-trip method invocation per second [17].

### Security

Refer to a previously given description. All the attributes of this element are required in this context.

### Backup

Refer to a previously given description. Only *solution*, *complete_backup_interval* and *data_type* attributes are required in this context.

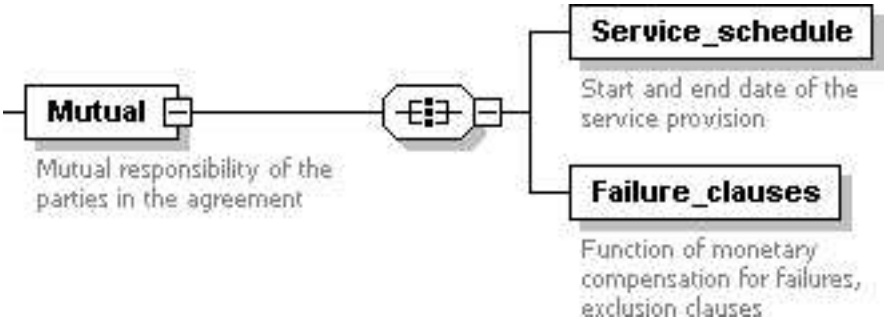



Figure 24: Mutual responsibilities in a *Hosting* SLA.

#### Monitoring

Refer to a previously given description. *Report_method*, *report_frequency* and *reporting_on_demand* attributes are required in this context.

### 4.6.3 Mutual responsibility

In Figure 24 a schema representation of mutual responsibility is given. We have discussed the elements there depicted and hence we do not describe them again (refer to preceding Sections as appropriate).

## 4.7 Service SLA

A *Service* SLA is contracted between application/service components providers (Figure 25).

### 4.7.1 Client responsibilities

#### Name

Name of the application/service component provider acting as a client in the agreement (Figure 26). It is a simple element, specified as a string.

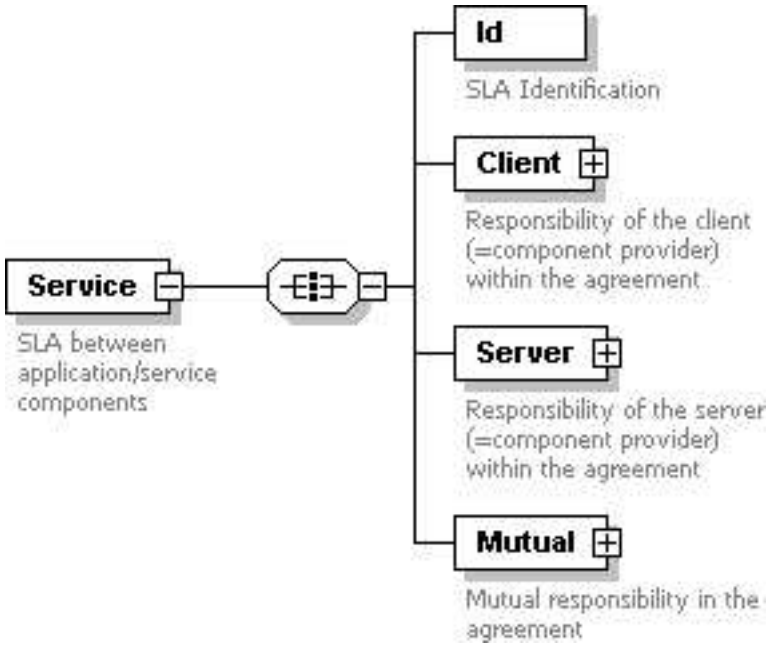



Figure 25: General structure for a *Service* SLA

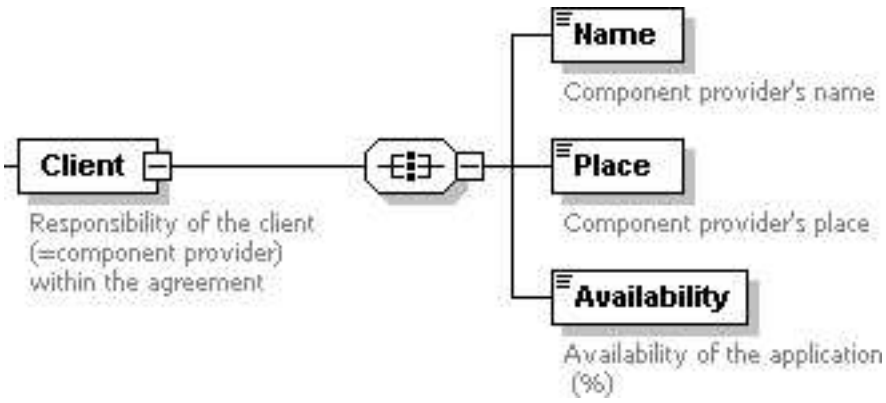



Figure 26: Client responsibilities in a *Service* SLA.

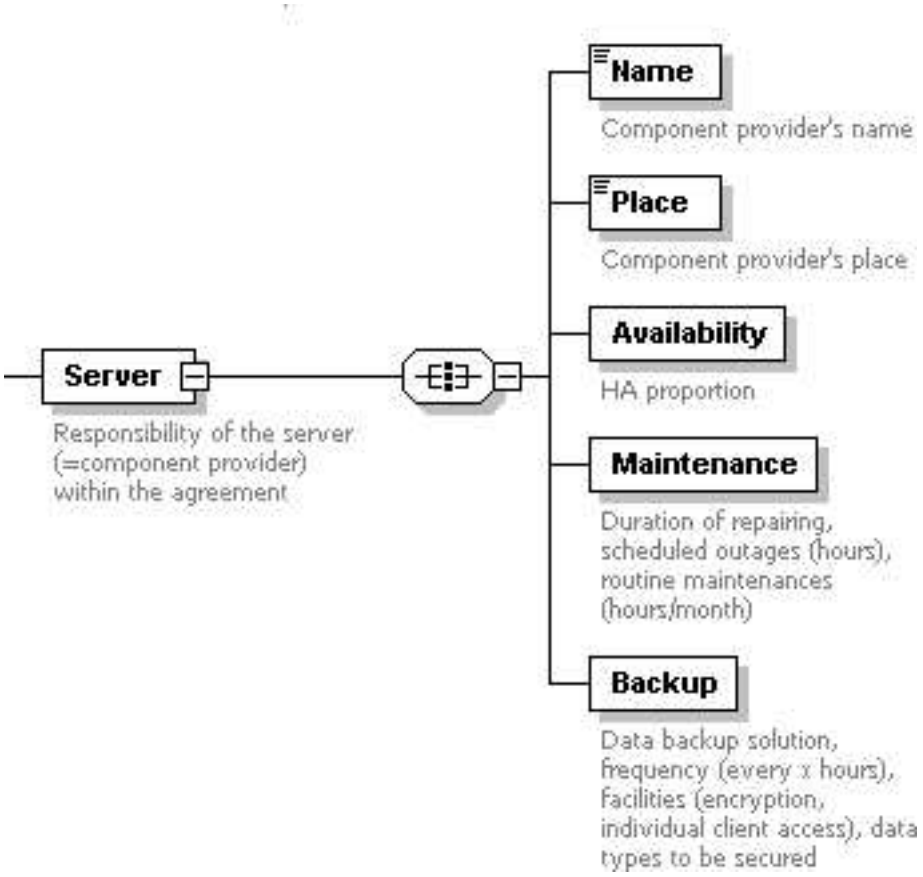



Figure 27: Server responsibilities in a *Service* SLA.

#### Place

Geographical location of the application/service component provider acting as a client in the agreement (Figure 26). It is a simple element, specified as a string.

#### Availability

Simple element expressing the percentage of the service-component availability over one year.

### 4.7.2 Server responsibility

#### Name

Name of the application/service component provider acting as a server in the agreement (Figure 27). It is a simple element, specified as a string.

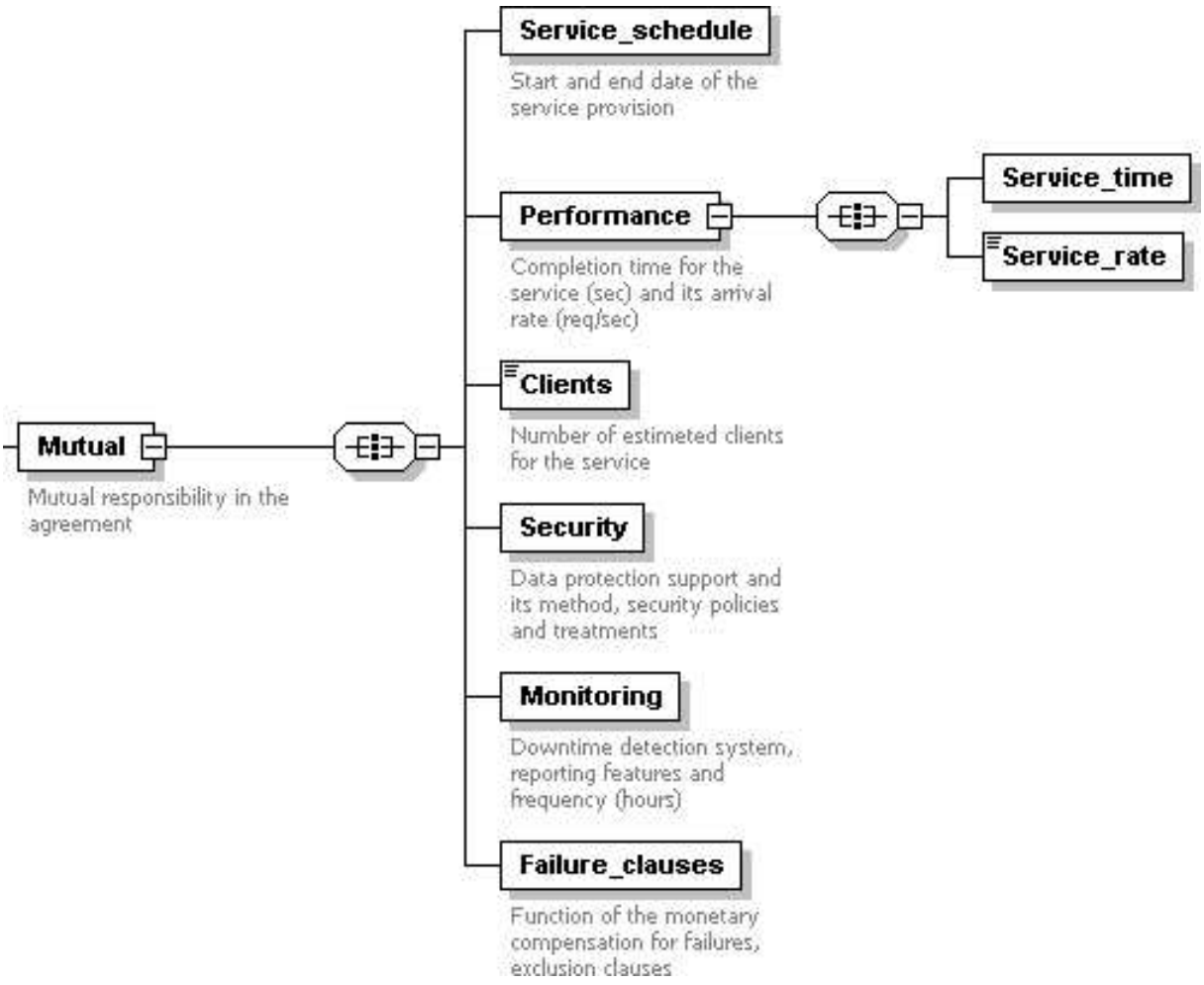



Figure 28: Mutual responsibilities in a *Service* SLA.

**Place**

Geographical location of the application/service component provider acting as a server in the agreement (Figure 27). It is a simple element, specified as a string.

For the definition of Availability, Maintenance and Backup refer to previous Sections as appropriate.

### 4.7.3 Mutual responsibility

For the definition of Service schedule, Security, Monitoring and Failure clauses refer to previous Sections as appropriate. For Performace and Clients refer to the following.

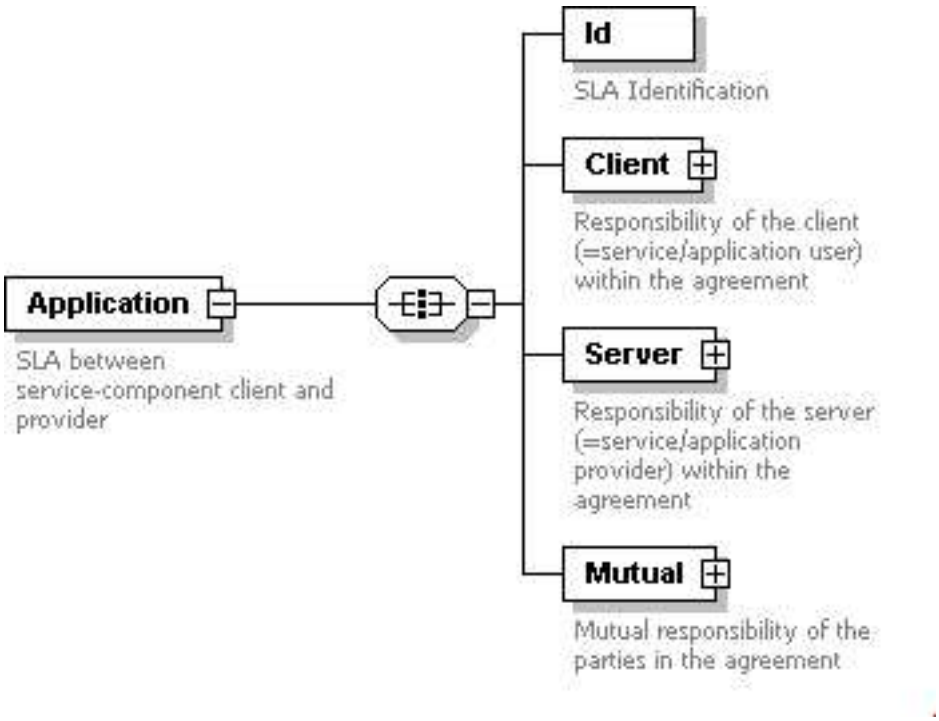



Figure 29: General structure for a *Application* SLA

### Performance

In this context, performance is defined by completion time for the service in seconds and its arrival rate in requests per seconds. Two sub-elements belongs to this element: Service time and Service rate, the former being a complex element. In fact, it is possible to specify average completion time (required) and optionally maximum, minimum and its distribution.

### Clients

Number of extimated clients for the service can be defined and agreed using this simple element.

## 4.8 Application SLA

A *Application* SLA is contracted between service-component client and provider (Figure 29).

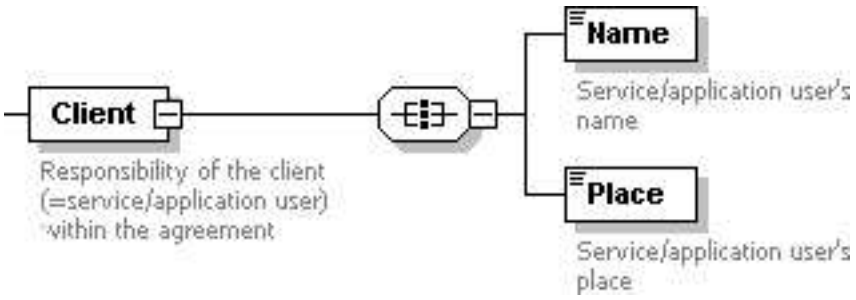



Figure 30: Client responsibilities in a *Application* SLA.

### 4.8.1 Client responsibilities

#### Name

Name of the service-component client (Figure 30). It is a simple element, specified as a string.

#### Place

Geographical location of the service-component client (Figure 30). It is a simple element, specified as a string.

### 4.8.2 Server responsibility

#### Name

Name of the service-component provider (Figure 31). It is a simple element, specified as a string.

#### Place

Geographical location of the serice-component provider (Figure 31). It is a simple element, specified as a string.

#### Availability

Simple element expressing the percentage of component availability over one year.

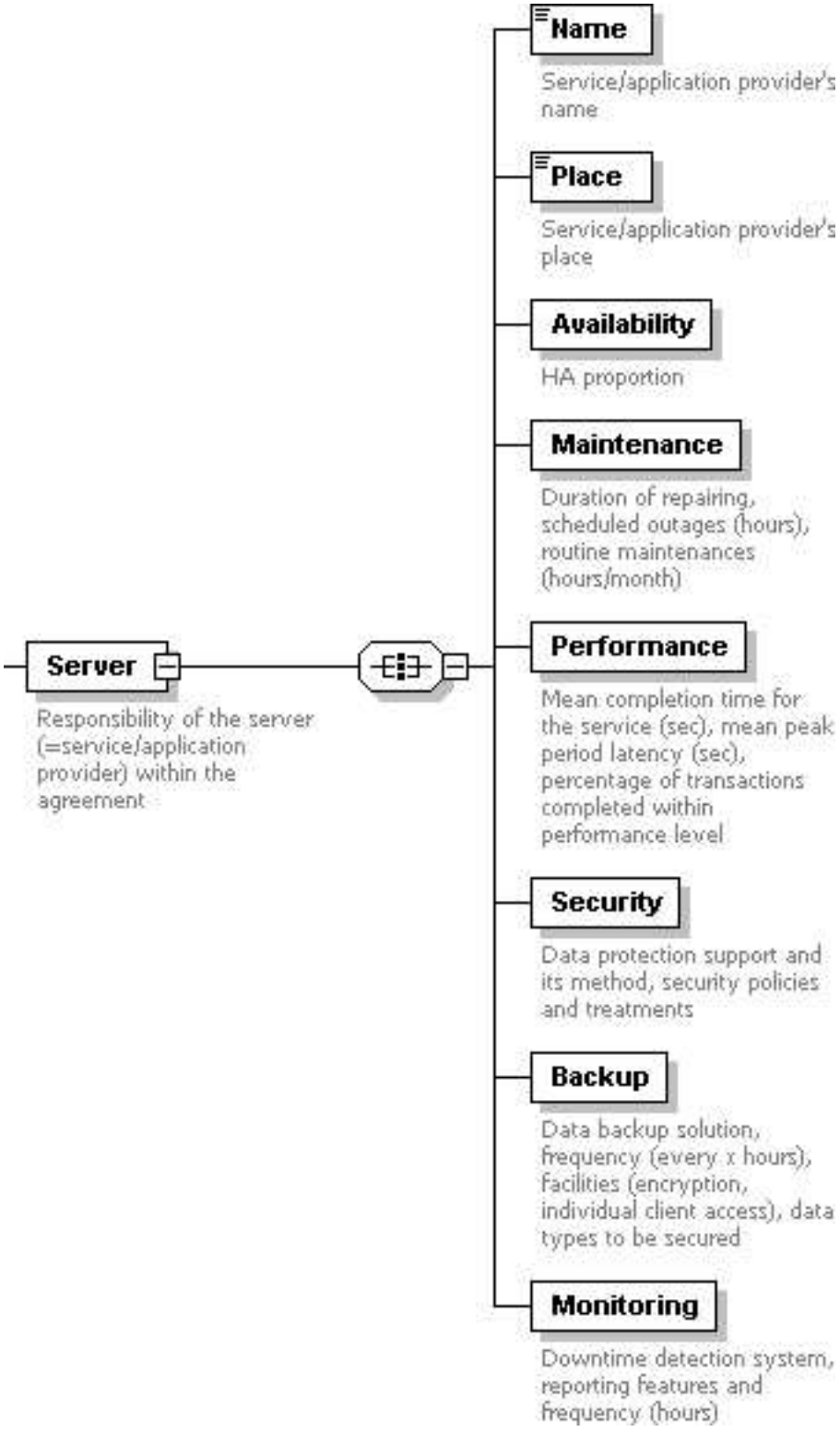



Figure 31: Server responsibilities in a *Application* SLA.

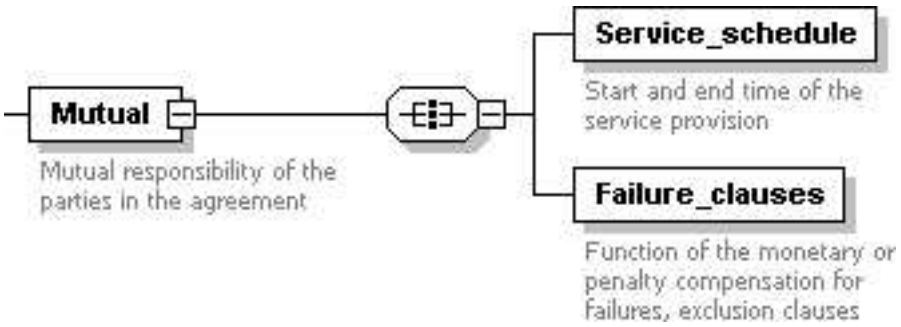



Figure 32: Mutual responsibilities in a *Application* SLA.

**Performance**

In this context, performance is defined by latency in seconds, peak time latency in seconds and percentage of transactions completed within performance level.

For the definition of Maintenance, Security, Backup and Monitoring refer to previous Sections as appropriate.

### 4.8.3 Mutual responsibility

In Figure 32 a schema representation of mutual responsibility is given. We have discussed the elements there depicted and hence we do not describe them again (refer to preceding Sections as appropriate).

# Chapter 5

# CPXe: a SLAng case study

## 5.1 Case study: CPXe

The Common Picture eXchange environment (CPXe) is an I3A[1] 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 33, 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 mini-lab applications.

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

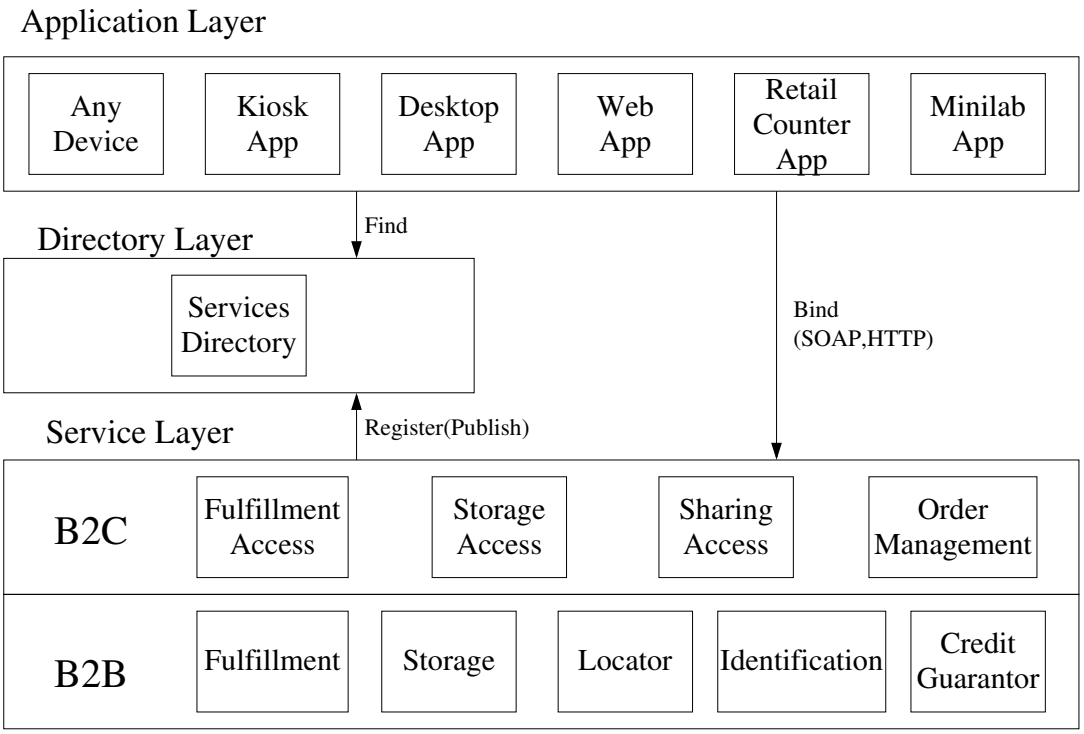



Figure 33: CPXe Architecture

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.

## 5.2 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.

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.

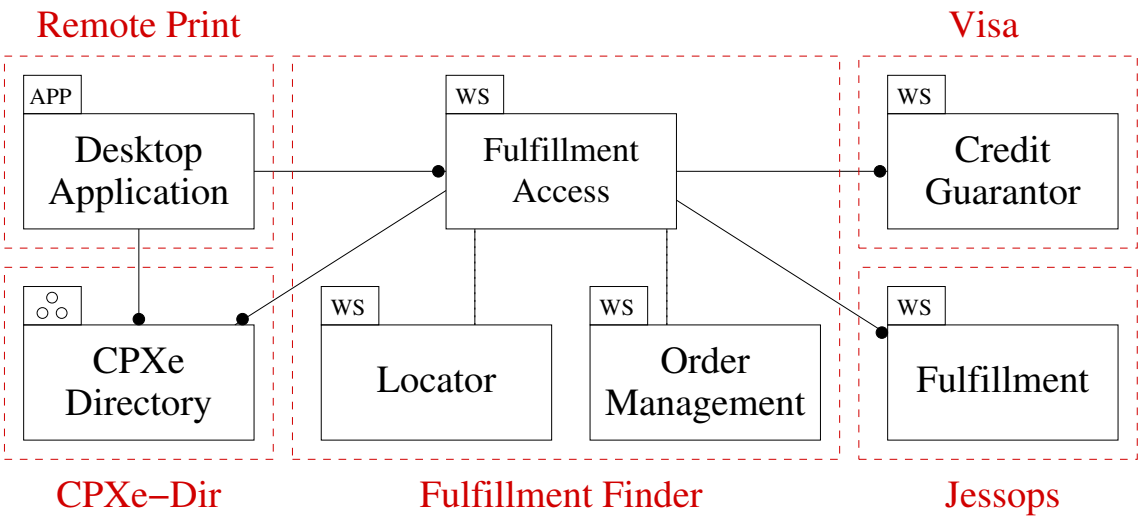



Figure 34: Scenario1: Print service from home.

## 5.3 Three scenarios

Three likely scenarios we have been analysed are now presented (Figures 34, 35 and 36). In order to describe service composition, we used our architectural notation (refer to Section 2.3). 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 a little identifier mark over their box.

Competence domains are delimited by a dashed line polygons. Every competence domain contains the entities 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 remind that SLAng-generated SLAs within the same business boundaries can help the process of internal service-composition modeling and external SLA-offering (Section 3.2). SLAs with a legal value (SLA contracts) are those which imply crossing domain boundaries.

### 5.3.1 Scenario 1: Print service from home

*Remote Print* is a desktop application to upload images for printing at print shops (Figure 34). It looks up into CPXe directory, listing CPXe Web Services. The

user of *Remote Print* chooses *Fulfillment Finder* Web Service, based on its better offer in terms of cost and service level. It offers mean completion time of 4 seconds and 93% of transaction completed within performance level. *Fulfillment Finder* locates print shops and handles order management. It uses an additional Web Service that apply business logic, consumer information and queries against the directory (smart locator).

Finally, *Jessops* is choosen as print shop, once again for cost, service level and other factors (e.g., it is near home). *Jessops* provides a specific Web Service for fulfillment that is used by *Fulfillment Finder* to handle the user request. Images can now be uploaded for printing under a specific service agreement (service completion time of 1 hour). Payment can be made using a Web Service provided by *Visa*, which validates transactions. *Visa* offers successful transaction ratio 99.2% and 99.9% of availability. Only *Application* and *Service* SLAs are represented in this scenario.

In Figure 34, only *Web Service* entities are depicted in some *Competence Domains*. This can mean that the underlying infrastructure for service provisioning (components, containers, databases) belongs to the same business or, even simplier, that we do not represent other possible business interactions, regulated by more SLAs. For example, no *Networking* SLAs are illustrated, even if they are, of course, in place.

Service composition allows us to focus only on service interfaces we are interested in and consider the others as given. This observation applies to all the scenarios in this paper.

### 5.3.2 Scenario 2: Picture access.

*Online Photolab* is an application suite intended for use in print shops (Figure 35). The retailer can connect and let his or her customers connect to a storage service. 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 let users discover CPXe services, finds

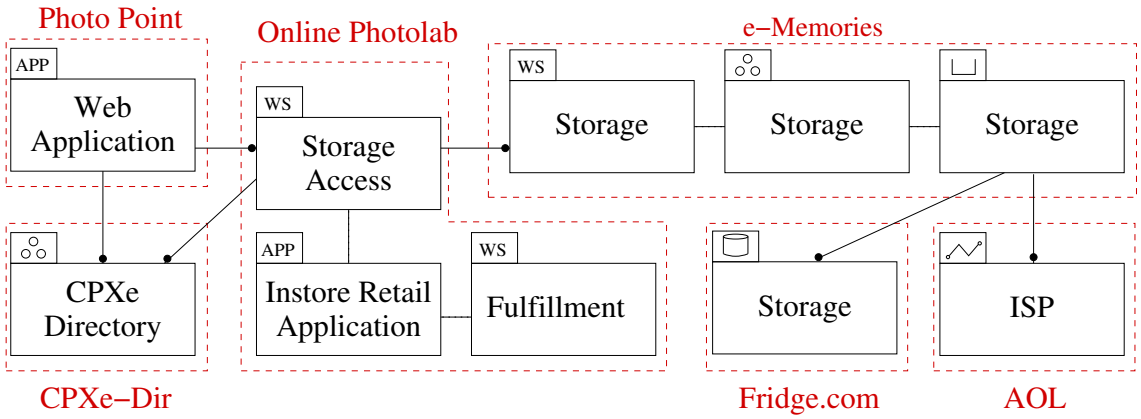



Figure 35: Scenario2: Picture access.

*Online Photolab* in *CPXe-Dir*. *Online Photolab* looks up in it as well to find 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 offers 99% of availability at a reasonable price.

Figure 35 shows, this time, a more detailed description of *e-Memories Competence Domain*, including internal relationships between Web Services, Components and Container. *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.

In the picture 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. The others are *Application* and *Service* SLAs.

### 5.3.3 Scenario 3: Upload/order from a kiosk.

*Every-ware* is an application for uploading images from a digital camera and ordering fulfillment services from a kiosk (Figure 36). Users can free their digital camera (e.g., while on holiday) and print photoes to the nearest print shop, or the nearest home one or even mail them. Looking up in *CPXe-Dir*, *Photo access*, a Web Service suite to locate and use storage services, order fulfillment and mail

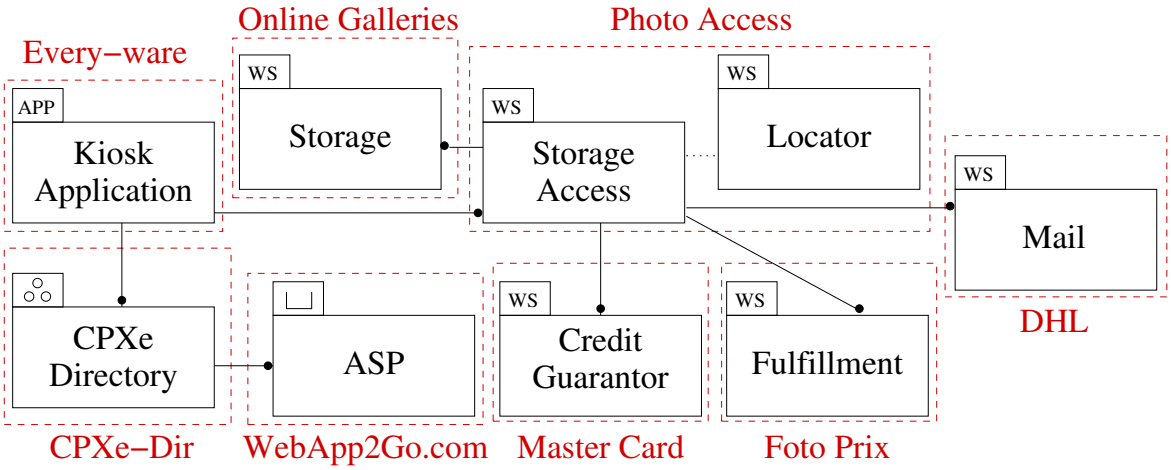



Figure 36: Scenario3: Upload/order from a kiosk.

services is found. It ensures less than 3% as aborted transaction ratio.

Using *Photo Access* locator, the user finds *Online Galleries*, which offers forty Megabytes of disk space and 99.8% of data integrity. The user further chooses to print at a local shop, *Foto Prix*, and send a copy home, via *DHL*. Transaction are guaranteed by *Master Card* Web Service.

This time CPXe Directory replication at *WebApp2Go.com* ASP is represented. Between them there is a *Hosting* SLA. *WebApp2Go.com* offers a EJB round-trip method invocation per sec of 50ms.

The other SLAs in this scenario are *Application* and *Service* SLAs.

## 5.4 SLAs for the scenarios

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 the previous scenario Figures are shown to convey the expressiveness of SLAng. Furthermore, we include an example of an SLA written in a natural language in Figure 41. By comparing it with the same SLA in SLAng language in Figure 39, one can gain insights into

SLAng characteristics and appraise its remarkable concision.

```
<?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>
```

Figure 37: *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>
```

Figure 38: *Persistence* SLA between a container provider and an SSP

```
<?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>
```

Figure 39: *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>
```

Figure 40: *Communication* SLA between container provider and an ISP

```
This Deed of Agreement is entered into as of the Effective Date identified below.

BETWEEN
WebApp2Go.com of London (To be known as the Server in this Agreement)
AND:
CPXe-Dir of Los Angeles (To be known as the Client in this agreement).

WHEREAS Server desires to enter into an agreement to supply Client with
Replication of software components (To be known as Hosting in this Agreement).
NOW IT IS HEREBY AGREED that Server and Client shall enter into an agreement subject to the following terms and conditions:

1. Definitions and Interpretations
1.1 All information (order, payment, notifications, etc.) is to be sent electronically.
1.2 This agreement is governed by British law and the parties hereby agree to submit to
    the jurisdiction of the Courts of the Britain with respect to this agreement.

2. Commencement and Completion
2.1 The commencement date is scheduled as the 11th of February 2003.
2.2 The completion date is scheduled as the 10th  of February 2004.
2.3 The schedule may be modified by agreement as defined in Section 9.

3. Service use
3.1 Client will allow a mean number of active service clients equal to 24,000, with a maximum peak of 40,000.
3.2 Client will guaranteed an arrival rate of service requests not greater than 113.2 per  second.

4. Provision
4.1 500 Mbytes of disk space will be provided.
4.2 High availability will be enabled, with 99.6% as the percentage of availability over a year and 2 hours as mean recovery time.
4.3 20 hours of scheduled outages and 12 hours of routine maintanance are planned.

5. Performance
5.1 Server guarantees a mean response time equal to 2.6 seconds, with peak time latency of 4.7 seconds.
5.2 Server guarantees a successful transaction ratio equal to 98% over a year.
5.3 The cluster throughput, based on 9 containers, with 310 active clients, will guarantees a method invocation rate of 53.141.

6. Security
6.1 Data protection will be enabled using RSA as the encryption method.
6.2 Certification, authentication and virus scanning will be guaranteed at any time.
6.3 User configuration data will be archived using Rar, and backed up using REOBack.
6.4 A complete backup wil be executed every 24 hours, with an incremental backup interval of 6 hours.
6.5 Users will be given the possibility to access individually their backup data.

7. Monitoring
7.1 Server will use IDX as a tracking system for monitoring performance levels.
7.2 Server will report every 48 hours the state of the system, using a properly formatted XML document.

8. Termination
8.1 If Client or Server fail to carry out any of its obligations and duties under this agreement
    the offender is to be considered in breach of the e-contract.

9. Disputes and compensation clauses
9.1 Server and Client  shall attempt to settle all disputes, claims or controversies arising under or in connection
    with the agreement through consultation and negotiations in good faith and a spirit of mutual cooperation.
9.2 Server and Client shall provide electronic evidences about breaches of the e-contract.
9.3 Unless it is proved that Client has caused outages, Server shall provide for a compensation equal to 4 times as much
    the difference between the promised availability and the delivered one.
9.4 This method of determination of any dispute is without prejudice to the right of any party to have the matter judicially
    determined by a British Court of competent jurisdiction.

10. Amendment
10.1 This agreement  may only be amended in writing signed by or on behalf of both parties.

E-SIGNATURES
In witness whereof Server and Client have caused this agreement to be entered into by their duly
authorized representatives as of the effective date written below.

Effective date of this agreement: 7th of February 2003
            [E-signature]                          [E-signature]
            [Person]                                 [Person]
            [Role]                                     [Role]
E-address for Notices:
            [E-address]                            [E-address]
```

Figure 41: *Hosting* SLA in a natural language (English).

# Chapter 6

# Evaluation and future work

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 [6]. 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.

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.

# Bibliography


[1] Selim Aissii, Pallavi Malu, and Krishnamurthy 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] Werner Beckman and Marina Oleneva. Application hosting requirements. Technical report, Adesso AG, OCT 2002.

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475 - An Architecture for Differentiated Services. *http://www.ietf.org/rfc/rfc2475.txt*, December 1998.

[5] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible markup language. *World Wide Web Consortium*, March 1998.

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

[7] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*, mit press edition, 2000.

[8] CPXe A consumer digital photography industry initiative. A business perspective white paper. Technical report, i3a International Imaging Industry Association, MAY 2000.

[9] CPXe A consumer digital photography industry initiative. A technical perspective white paper. Technical report, i3a International Imaging Industry Association, JUN 2002.

[10] Wolfgang Emmerich. Software engineering and middleware: a roadmap. *International Conference on Software Engineering ACM Press New York, NY, USA*, 2000. Limerick, Ireland.

[11] S. Salsano et al. Definition and usage of slss in the aquila consortium, November 2000.

[12] David C. Fallside. XML Schema. Technical Report http://www.w3.org/TR/xmlschema-0/, World Wide Web Consortium, APR 2000.

[13] 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.

[14] 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.

[15] M. Hondo, N. Nagaratnam, and A. Nadalin. Securing web services. *IBM Systems Journal*, 41(2), 2002. New Developments in Web Services and E-Commerce.

[16] 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), jun 2001.

[17] Sacha Labourey and Bill Burke. Jboss clustering. Technical Report http://www.jboss.org, JBoss Group, LLC, jun 2002.

[18] D. Davide Lamanna, J. Skene, and W. Emmerich. SLAng: A language for defining Service Level Agreement. In *IEEE Computer Science Press*, December 2002. Accepted for publication at FTDCS 2003.

[19] 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), apr 1998. Kyoto, Japan.

[20] Jim Martin and Arne Nilsson. On service level agreements for ip networks. *IEEE Infocom*, jun 2002. New York, USA.

[21] Hermann De Meer, Wolfgang Emmerich, Cecilia Mascolo, Nicola Pezzi, Miguel Rio, and Luca Zanolin. Middleware and management support for programmable qos-network architectures. *Presented at IWAN'2001*, October 2001. Philadelphia.

[22] George Memenios, George Pavlou, David Griffin, and Leonidas Georgiadis. Service level specification semantics and parameters. Internet Draft, tequila-sls-02, February 2002.

[23] T. Mikalsen, I. Rouvellou, and S. Tai. Reliability of composed web services from object transactions to web transactions. Technical report, IBM T.J. Watson Research Center, 2001. New York, USA. Proposal submitted to the OOPSLA 2001 Workshop on Object-Oriented Web Services.

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

[25] IST Project. Creation and deployment of end-user services in premium ip networks(cadenus), 2002. http://www.cadenus.org.

[26] Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers (standards track). RFC 2474.

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

[28] 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.

[29] R. Sprenkels, A. Pras, B. van Beijnum, and L. de Goede. A customer service management architecture for the internet. *11th IFIP/IEEE DSOM 2000*, 2000.

[30] Scott Stark and Marc Fleury. Jboss administration and development. Technical Report http://www.jboss.org, JBoss Group, LLC, jan 2002.

[31] Zheng Wang. *Internet QoS: Architectures and Mechanisms for Quality of Service.* Morgan Kaufmann Publishers, harcover edition, March 2001. ISBN: 1558606084.