# Software Architectures for Service Level Agreements and Contracts

Carlos Molina-Jimenez[1], Jim Pruyne[2], and Aad van Moorsel[1]

[1] University of Newcastle Upon Tyne
School of Computing Science
Newcastle upon Tyne, NE1 7RU, United Kingdom
{carlos.molina aad.vanmoorsel}@ncl.ac.uk
[2] Hewlett-Packard Laboratories
1501 Page Mill Rd.
Palo Alto, CA 94304
jim.pruyne@hp.com

**Abstract.** Service level agreements have traditionally played an important role in computing system management solutions. With the emergence of business-to-business, utility and grid computing, additional types of contracts and agreements have emerged. These agreements serve several purposes, from documenting business agreements to automating security and service management, and come in various shapes and forms, from paper copies to software objects. It is being realised by experts in the field that in order to support automated adaptation and management of systems and operations, agreements should be encoded and integrated in software platforms that support multi-party interactions and/or management solutions. In this paper, we review the state of the art in software architectural support for various forms of agreements, for all stages of their life-cycle. We also review emerging platforms in standard bodies, industries and academia.

## 1 Introduction

We will argue and illustrate in this paper that a distributed computing infrastructure must incorporate *agreements* as first-class software building blocks if it wants to support automated adaptation and management in the presence of multiple (possibly competing) interests. These agreements represent expectations and obligations of various partners about the functionality and performance of systems and services. But more so, these agreements are a tool to enable automated decision-making for adaptation and management, and can be useful even if there are not multiple parties involved in formulating the agreements (as we will make clear below).

Modern-day and emerging computing infrastructures are increasingly flexible in their support of computational models as well as business models, as witnessed by the developments of adaptive and on-demand computing solutions proposed by the likes of HP, IBM, Oracle and SUN. These solutions typically envision a service provider model for various aspects of computing, such as CPU use, network use, application hosting, etc. From a software platform perspective, such solutions are often tied to the grid, which deals with resource sharing across multiple parties using open software. The solutions rely on virtualisation of resources and applications, shielding the customer or operator from the complexities of the underlying infrastructure.

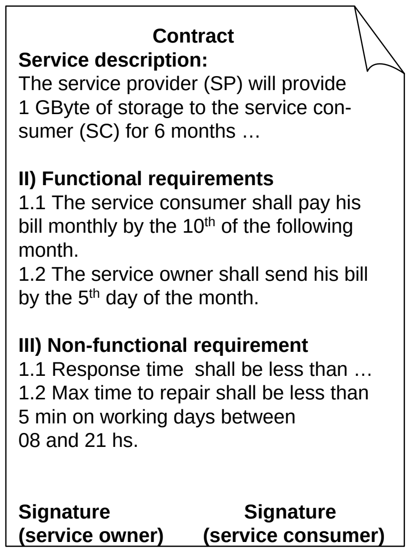
It will be interesting to see how the various research and commercial solutions will fare over the coming years and decades, and how the market judges the trade-off between added functionality and increased complexity. Irrespective of how things develop, it seems to us that a necessary piece of any infrastructure supporting such computing models must be an embodiment of 'agreements', which the platform uses if some of its functions require to know the expectations and obligations of involved physical and logical entities. Agreements come at various levels of granularity, ranging from service level agreements as used in the telecommunication industry, to full-blown contracts between business-to-business partners that include clauses that stipulate how and when purchase orders must be processes, the average response time of the system, etc. Agreements naturally fit the service provider model, but also provide the necessary information to allow for automated decision-making by self-managing components and services.

There are many open issues in the technology required to support agreements. The emphasis in this paper is on a survey of existing and ongoing work, with pointers to remaining research issues. We divide up the technologies in (1) specification, (2) provision, (3) monitoring, (4) adaptation and (5) resolution, roughly following the life cycle of typical agreements. Table 1 summarizes our findings. In Section 4 we then discuss proposed solutions by standards bodies (WS Agreement in the Global Grid Forum), industries (Hewlett Packard and IBM enterprise IT) and academia (TAPAS, an EU-sponsored research project). To set the stage, we now first discuss terminology used in this paper and in the literature.

## 2 Usage Models and Definition of Agreements

**Contracts.** In the business model assumed in this interacting parties (for example two business enterprises that decide to trade over the Internet) are mutually suspicious and reluctant to engage in business interactions unguarded, because of this, interactions are regulated by legal contracts signed by the interacting parties. Contracts come in different forms, sizes and shapes, yet in our view contracts used to regulate Internet interactions should have the following three elements: header, functional and non–functional requirements (see Fig. 1); it should be noted that authors that address the issue of contract automation, call these requirements functional and non–functional Service Level Agreements (SLAs), respectively.

The contract header consists of names and addresses of the parties involved, star and end dates and a description of the service in English prose. *Functional*



**Contract**
**Service description:**
The service provider (SP) will provide 1 GByte of storage to the service consumer (SC) for 6 months …

**II) Functional requirements**
1.1 The service consumer shall pay his bill monthly by the 10th of the following month.
1.2 The service owner shall send his bill by the 5th day of the month.

**III) Non-functional requirement**
1.1 Response time shall be less than …
1.2 Max time to repair shall be less than 5 min on working days between 08 and 21 hs.

**Signature (service owner)** **Signature (service consumer)**

**Fig. 1.** Elements of a typical contract for Internet business.

*requirements* encompass terms and conditions that stipulate how high level business interactions such as *request purchase order*, *send payment, send refund*, etc. On the other hand, *non-functional requirements* include technical parameters that describe the performance of a system, such as *response time, availability, reliability, time to repair*, etc. Intuitively, functional requirements are meant to describe the reliability of a service that a business partner offers such us book selling and ticket reservation, where the input and output that the service accepts and receives depend of decisions made by the humans that run the service. Likewise, non–functional requirements are meant to describe requirements that are not influenced by the humans that run the service at will.

**Usage Models.** There exists a progression of usage models for agreements. First, we have what is known as *contract management*, in this model an agreement is a computer readable file, or a set of them, that describes a deal between a customer and a provider. At the least, the electronic agreements serve the purpose of book keeping, both for customers and providers.

Secondly, an obvious next step is to migrate from readable files into executable files; that is, to develop software sophisticated enough to automatically or semi-automatically monitor the observance of the agreement to allow the involved parties to determine (possibly at run–time) if the agreement is met. Such usage of agreements is typical for service level agreement deployment in telecommunications, we will call it *automated agreement monitoring*.

Thirdly, we can further automate the process of initiating and adapting service usage, as a matter of protocol, so that all changes in the platform follow specification of the agreement, which will normally identify elements such as access rights, quality–of–service expectations, penalties, etc. We will call this approach *automated agreement management.* With this approach, the platform then effectively utilises agreements as the core element to keep itself organised, and communicate organisational aspects to the involved parties. Moreover, it inspects the agreements continuously if access rights or other operational rights and obligations need to be assessed.

Finally, an agreement can be the core piece of information required by self-managing components, optimisation algorithms, and other decision-making entities to enable them to automatically adapt and manage systems and services. We call this usage model *automated internal agreement-based management.*

In the last two usage models, agreements do not require the explicit involvement and agreement of multiple parties. Instead, they can be introduced by the management system as a set of goals and rules representing conflicting interests, but remain hidden to the interested parties. Various modules in the system use these 'hidden' agreements [24] for the purpose of automated adaptation and management.

**Terminology.** In our view, the usage model that has the largest implications on software architecture is the automated agreement management; for this reason, it will be primary focus of interest of this paper. In this paper, we therefore use the term agreement to mean:

> An *agreement* is a machine interpretable specification of the value of a set of selected parameters of a service instance, involving more than one (logical) party, to assist in automation.

We do not restrict in any way the chosen parameters, which can represent quality of service levels, penalties and payments, functional agreements about message exchanges, and anything else business partners may like to specify. Note, however, that specifications of messages exchanges, such as defined in BPEL (Business Process Execution Language) or RosettaNet, do not satisfy our definition of agreement, since the purpose of these standards is to tell designers of cooperative business applications what messages are involved in each business operation rather than defining how their execution environments are expected to perform upon sending or receiving messages; consequently, neither BPEL or RosettaNet have means of expressing business interactions as parameter-value pairs.

Neither do we restrict the type of automation one can think off, e.g., automated conformance monitoring, automated adaptation or management. Important in the above definition is that more than one party is involved, even if there is not necessary a physical or human embodiment of these parties, or even if the physical or human embodiments may not be aware of the agreements.

| | status | open or underdeveloped issues |
|---|---|---|
| Specification | various specification languages [14,38,11,13,40]<br>WSLA [11] in WS Agreement [2] | right mix of expressiveness<br>and simplicity standardisation<br>(domain-specific) ontologies |
| Provision | job scheduling<br>resource provision [16]<br>monitoring provision [38,11] | mapping between agreement<br>goals/constraints and resources<br>domain or product-specific<br>deployment templates [16] |
| Monitoring | automated monitoring at points-of-presence [26]<br>trusted third-party solution | measurement protocols [24],<br>enhanced trust solutions |
| Adaptation | optimisation algorithms [29,39]<br>self-management [18] | guaranteed Internet QoS,<br>self-management in grid<br>software [19,44] |
| Resolution | non-repudiation [9]<br>trusted third party | changing and terminating<br>agreements,<br>enhanced trust solutions |

**Table 1.** Survey of state of the art and open issues.

It may be good to compare our definition of agreement with existing ones, for instance that for service level agreement in IETF RFC 3198 [46]. In this work, a service level agreement is defined as ‘the documented result of a negotiation between a customer and a provider of a service, that specifies levels of QoS or other attributes of the service.’ A service level objective is ’a set of parameters and their value.’ This is a very natural definition, similar to ours, but different in the fact that it assumes parties to negotiate, a requirement no longer practical or desirable for our usage models. Note that the RFC 3198 definition also points to QoS, which is a typical connotation when SLAs are concerned.

It is important to clarify the relationship between policies and agreements. Agreements are certainly not policy rules, which IETF [46] defines as ‘a set of rules to administer, manage and control access to [...] resources.’ Instead, agreements are a multi-party variation of ‘policy goals,’ defined as ’a definite goal [...] to guide and determine present and future decisions.’ In our set-up, policy rules are one of many possible approaches to adapt and manage a system. Other possibilities include games, auctions or run-time mathematical optimisation algorithms. In summary, policies are excluded from our definition of agreement simply because we are interested in applications that entail business collaboration between two or more independent and autonomous enterprises.

## 3 Agreement and Infrastructure Requirements and Properties

To allow an infrastructure to automate adaptation and management based on agreements, the agreements must have various properties, and must enable various specific tasks at various stages of the agreement’s life cycle. We divide and discuss these properties and requirements in five groups, roughly corresponding to various stages in the life-cycle of an agreement.

1. **Specification.** Languages and formalisms for expressing the agreement, choice of metrics, desired/required contents, validity and changes, negotiation.
2. **Provision.** Automated and customised deployment of resources and monitoring tools, dealing with resource scarceness.
3. **Monitoring.** Techniques and tools for collecting performance metrics, algorithms for evaluation of performance, violation notification, functional and non-functional SLAs, third-party involvement, exchange protocols.
4. **Adaptation.** Decision-making about new requests and adapting resource allocations, business-driven, alarm handling, self-management and autonomic management.
5. **Resolution.** Auditing and non-repudiation, conflict resolution.

Within each item, we discuss issues ranging from security requirements and automation needs to standardisation and third-party solutions. Table 1 summarizes our findings.

## 3.1 Specification

To understand what a language for SLAs specification does it is perhaps useful to compare it against a service description language. As it names implies a service description language, such as WSDL (Web Service Description language) describes what the service can do and how the client can invoke it. The description of the service in a language that the two interacting parties understand ensure that two parties can validate and interpret each of the fields contained in the messages or documents that they exchange. However, a service description language is not meant to describe the behaviour of the interacting parties in terms of performance, that it, it defines, for example, what a server is expected to do, but it does not say how well, fast, efficiently, etc. This is where SLA languages play their role. The goal of a SLA is to describe the performance of the two interacting parties when they interact with each other, normally following a service description, to perform business. In our view, the service description and the SLA description are two issues that should be addressed independently; there is a large class of applications that in order to reduce costs, might opt for a best–effort service (one without any SLA expectations).

Because SLAs is not a well–understood issue yet, current Internet business contracts often leave computation and communication requirements unspecified and open to interpretation; for example, it is common to come across services providers advertising services whose availability is no less than 99.9% or average response time less than 2 seconds; specifications like these give the customer only a vague idea about what performance to expect from the provider; since the customer's interpretation can differ from the provider's, this can easily drive the business partners into disputes.

Common sense tells us that disputes can be prevented from happening or be fairly settled if one can find a means for describing SLAs in a notation that the two business partners (and possibly third party observers) read and interpret in the same way. We mention third parties here because we envision that some business partners might rely on third parties to conduct the required monitoring activities. Likewise, in disputable situations caused by the occurrence of failures, third parties (for example, an arbitrator) might be called to analyse records of evidence about the business interaction and settle a dispute. The challenge here is to find such a language. The issue has been studied both by academic and industry researchers; as a result several proposals have emerged; in the following sections we will discuss what we consider the most representatives in terms of originality and commercial influence; our hope if that our unbiased criticism will give the reader a good understanding about what is provided and missing in each proposal.

To put our discussion in context it is perhaps useful to clarify that in general the proposals for SLAs specifications can be classified into two broad categories:

**Ontological specifications:** They rely on the expressivity of natural languages to describe the behaviour of a contracting party; thus an ontological specification for response time will look more or less like this: "The response time for operation $O_i$ is the amount of time, in seconds, elapsed between the start of the operation and the receipt of the expected results". In our opinion, ontological specification are of little use in automatic monitoring and enforcement of contractual interaction, unless they are supported by some means of formalism; for this reason we will not discuss them further.

**Formal specifications:** They rely on the power of formal notation to describe the behaviour of a contracting party; rather that using elaborate English sentences, a formal notation uses a precise and compact mathematical notation, for example, the response time for an operation will be described by the algorithm or arithmetic equation to compute it.

Another comment that might help the reader to follow the discussion is that specification of SLAs is still a research topic; consequently, there is consensus yet about what QoS parameters are the most relevant to assess the QoS offered by service providers; no surprisingly, different SLAs languages offer means for describing different parameters; all depends on what their designer had in mind at design time.

**SLAng** SLAng is a language for specifying non–functional SLAs expected between two entities that offer some service to each other. Unlike similar languages that focus only on interactions at Web services level, SLAng was designed with generality in mind, thus it can be used to specify SLA for application service provisioning, Internet service provisioning, storage service provisioning, and other services, regardless of the level at which the interaction occurs.

The non–functional SLAs are specified in what is called, a SLAng document. From the point of view of its contents, a SLAng document is contractual document between two parties and contains the usual contract headers (names, addresses, start and end dates, etc.) and the service level specification (SLS) that rigorously specify the expected performance of the interacting parties. In SLAng, reduction of ambiguity of SLAs is achieved by means of applying meta–model techniques. First, the syntax of SLAng is modelled in UML (after a manual translation from XML), this defines the format of SLAs. Secondly, the abstract syntax model is embedded in what is known an environment and behaviour model. Intuitively speaking, this environment and behaviour model defines the semantic of SLAng, as it describes the service infrastructure (objects) and the events (for example, response time less or equal $t$) associated with the behaviour of both the provider and the consumer. The violation of the semantics is captured by introducing OCL (Object Constraint Language) constrains into the model. The goal of co–locating the syntactic and semantic models is to associate syntactic and semantic elements so that the scope for disagreements over the meaning of syntactic terms is narrowed. The strong side of SLAng is that is a formal specification language which leaves little or no room for ambiguities is SLAs specification. Another positive aspect of SLAng is that it is an XLM document that can be easily integrated with existing service description languages such as WSDL and BPL. Furthermore, the SLAng document can be edited and validated against language specification, using standard tools. The weak side of SLAng is that it is still an on–going work, consequently, it latest version do not incorporate yet crucial parameters such as management actions (penalties) to respond to SLAs violations; it does not include either SLAs templates or reuse of SLAs clauses, consequently, it does not have means for reusing parts of specifications between classes of services or from developing new specifications from existing ones; perhaps the most serious limitation of SLAng is its lack of flexibility in QoS metric definition as in SLAng these metrics are built into the language, unlike WSOL where they are defined externally by means of an ontological vocabulary.

**The Web Service Offering Language** The Web Service Offering Language (WSOL) is an academic proposal for specifying SLAs, designed at Carlenton University [42]. WSOL focuses on Web service interactions. One of the strengths of WSOL is that is an XML notation fully compatible with the WSDL description language. As its names suggests, WSOL extends WSDL with capabilities relevant to service offering. A key concept in WSOL is that of **classes of services** which are defined as services of the same functionality but with different constraints; the goal behind this concept is to cater for customers with different budgets and needs. Thus the authors of WSOL envision that an arbitrary Web service will offer a service $S$ as a set of classes $c_1, c_2, ..., c_m$, where $c_i$ and $c_j$ offer the same functionality but differ in terms of constraint parameters that can be formally expressed as Boolean or arithmetic expressions; the use of formalism for expressing constraints results in a formal specification of classes of services. An important particularity of WSOL is that it enables the specification of a wide range of constraints; their designers claim that its latest version can specify functional and non–functional constraints, simple access rights, payment (for service and penalties), contractual parties (provider and consumer) and monitoring third parties, and relationships between service offering. Another valuable particularity of WSOL is its emphasis on reusability, that is, it has means for using a given specification for a class of service several times; this simplifies the specification of new classes of services and simplifies monitoring procedures.

**IBM's Web Service Level Agreement Language** The IBM's Web Service Level Agreement Language (WSLA) is a SLAs specification language [23]. As its name suggests, the focus of the WSLA is Web services, that is, application interactions; however, it can also be used to specify SLAs between two interacting parties lower levels. A SLAs specification written in WSLA is an XML document that formally defines what the two interacting parties expect from each other in terms of response time, throughput and similar parameters; it contains three major parts: the names of the parties (signatories and supporting third parties), the service definition (valid operations, metrics, parameters, measurement), obligations (threshold to be met) and penalties and administration procedures to activate when SLAs violations are detected. The goal of an WSLA specification is twofold; at deployment time it helps the interacting parties to configure their resources so that the SLAs at met at run time. At run time it helps the interacting parties to monitor each other performance and to detect and notify violations. Unlike WSOL, WSLA does not support the concept of service classes; this miss can be serious limitation in Web service applications that sell services to several customers as the management of such applications can become rather complex. Like in SLAng and unlike in WSOL, QoS metrics in WSAL are defined within the model.

### 3.2 Provision

As we have mentioned repeatedly, the main point of introducing electronic agreements is to automate various aspect of system and service management and adaptation. In this section we consider the first phase of the agreement life cycle, namely the provision of the service specified in the agreement as well as the provision of the monitoring software to verify if an agreement is being met.

Since we are dealing with contracting parties that are mutually suspicious, the main problem here is to deploy the necessary infrastructure to monitor the observance of the contractual agreements. For instance, in automated agreement management, the contracting parties need to deploy their executable version of the original contract to ensure that the observance of functional requirements is monitored; however, this is not straightforward as the formal model of the functional requirements needs to be checked for logical consistency before creating its executable version, to guarantee that the latter offers an acceptable level of dependability. The goal here is to remove possible faults that the original and informal specification of functional requirements might have as a results of errors made by the humans in charge of drawing-up the contract. Secondly, the executable code of the functional requirements needs run time support; it needs a mechanism to collect evidence about the business interaction and to drive the executable code back to synchrony when failures in the execution environment such as unexpected delays and network breakdowns drive the executable model into inconsistent states.

In terms of the software architecture we can identify two extremes in the range of possible approaches. On one end, the functional specification of a service interaction, or of resource usage, is given in the form of a file, typically human readable for reasons of convenience. Examples of this are job descriptions using the Grid job description language, or utility computing definitions in the SmartFrog language [16]. At the other extreme, the functional specification is a first-class software object (or service) itself, with standardised interfaces. This is the idea behind WS Agreement, which we discuss in detail in Section 4.1. The mixture of the two is very natural, such as in the case of SmartFrog. A readable template-based resource specification is created and then interpreted by the SmartFrog provision system, which keeps track of it as Java object.

When we review the technologies of HP and IBM in Section 4, we see these are driven by utility or on-demand computing opportunities [20]. The key enabler of utility computing is software for automated provision, such as SmartFrog [16] and Oceano [3]. Enrichment of such software to be governed by agreements is needed to deliver on the promise of fully automated management in multi-party setting. As an aside comment, it worth mentioning that to the best of our knowledge Oceano is one of the first projects to address the issue of meeting SLAs without over provisioning, it was developed by IBM in 2001.

**Monitoring provision.** Much of industry research in the area of monitoring has been concerned with automatically igniting monitor activities, that is, with automated provision of monitoring. This emphasis is understandable, since one of the major challenges in using monitoring software is to minimise the effort required to instrument systems and initiate the monitoring.

A well-specified agreement is an excellent tool to determine what monitoring is needed. The challenge is not to specify the required monitoring, but to infer from the functional and service levels what monitoring should be started up, who is in charge of the monitoring, when alarms should be generated and when SLA breaches occur. This idea is described by Sahai *et. al.* [38,24] in the context of web services.

## 3.3 Monitoring

Monitoring is all about observing the behaviour of a business partner during the business interaction to collect non-repudiable evidence that can be used to:

- detect diversion from acceptable behaviour.
- trigger corrective mechanisms.
- solve disputes originated from failures to deliver the expected service.

In Section **??** we argued that most contract used in Internet business should contain functional and non–functional requirements. In the following subsections we will discuss the infrastructure to monitor this two kind of requirements. Since the focus of this paper is on automated agreement management, we will devote our effort to monitoring of non–functional; however, to help the reader put the issue in context, we will also briefly discuss monitoring of functional SLAs.

It is worth clarifying that the goal of both functional and non–functional requirement monitoring is to ensure that the interaction between the business partners progress as specified by the SLAs stipulated in the business contract; consequently, in practice, the contract itself will stipulate what is to be monitored, when, where and by whom. In addition to these contractual monitoring, it is sensible to expect that some business partners might deploy private monitoring infrastructure to conduct internal monitoring, that is, monitoring of their own resources perhaps with the intention of taking corrective measures before the contractual monitors detect and notify a SLAs violation; or perhaps with the intention of optimising the usage of their resources.

Information collected from both functional and non–functional monitoring are essential to business interactions based on SLAs as they serve two purposes:

– they are used as inputs to algorithms that compute performance and detect and notify diversion from acceptable behaviour.
– they are used as evidence to resolve possible disputes between the business partners about failures to meet expected behaviour.

From the above discussion it follows that data collected by monitors has to be non–repudiable; because of this, in several practical applications it is collected by third parties trusted by the business partners.

**Monitoring of functional SLAs** To monitor functional SLAs we need an infrastructure with focus on observing the sporadic occurrence or absence of specific events of interest such as *purchase order requested*, *payment send overrun its deadline*, *refund request*, etc. A functional requirement monitor is normally realised as a piece of software conceptually located between the two interacting parties, to intercepts the messages related to their business interaction. This piece of software is known as *executable electronic contract* (x–contract) as it contains all the necessary information to determine what messages are valid at a given step during the business conversation. Valid messages intercepted by the executable electronic contract are forwarded to its intended destination, whereas invalid ones are bounced back to the sender or ignored. A rather abstract and simplified infrastructure for monitoring functional SLAs is shown in Fig. 2.

How the electronic executable contract can be realised is a matter of preferences. Alternatives for formally representing the functional SLAs include Finite State Machines, Petri Nets, event calculus and other event–condition–actions mechanisms and temporal logics. Such electronic executable contract can be implemented centralised or distributed; furthermore, it can be deployed within the interacting parties or in a trusted third party that plays the role of a central authority.

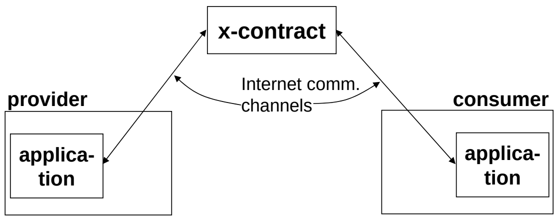


**Fig. 2.** Monitoring of functional SLAs.

**Monitoring of non–functional SLAs** To monitor non–functional SLAs we need an infrastructure with emphasis in observing and measuring the performance and availability of individual resources; examples of individual resources are cpu, databases, network, etc.; consequently, the monitoring system should collect enough information to compute response time, number of served transactions, time to repair, bandwidth, number of message lost, etc. With non–functional monitoring, the interest lies in detecting performance and availabity delivered below or above acceptable water marks; in practice, these water marks are evaluated over specific periods of time (for example, time to repair should be less than 24 hs) or over an specific number of operations (for example, 98% of all transaction should complete successfully). From this discussion it follows that practical realisations of non–functional monitoring systems include hardware or software–built sniffing sensors, interceptors and other similar devices that permanently, periodically or sporadically measure the device of interest and collects metrics about its performance.

Monitoring of non–functional SLAs has been discussed in several papers published in proceedings of conferences on e–commerce, Grid computing and Web services conferences; however, in most of these works the discussion of monitoring of non–functional SLAs is often mixed with Web and Grid services implementation technologies; these details in our opinion make it difficult to identify, isolate and reason about the key issues that monitoring of non–functional SLAs involves; a publication discusses the fundamental concepts of non–functional monitoring of SLAs at an abstract level is [25]; we will take this work as the basis for the discussion of this section; accordingly with the author of this work, the main issues the designer of system for monitoring non–functional SLAs has to keep in mind are: specification of SLAs, computational infrastructure used by the provider to build his service, communication infrastructure used by the consumer to gain access to the service, service point of presence, metric collection, measurement, evaluation and violation detection services.

**Computational and communication subsystem** The computation subsystem consists of the infrastructure (computers, LANs, databases, etc.) that the provider uses to produce the service before exposing it to the service consumers through one or more interfaces. The communication subsystem or infrastructure consists of one or more independent and autonomous ISPs that compose the communication path (e.g. $ISP_1$, $ISP_2$, $ISP_3$) to deliver the service from the provider's interface to the service consumer. In this scenario, the QoS received by a given service consumer is affected by both, the QoS guaranteed by the computation subsystem and the QoS guaranteed by he communication subsystem; a factor to keep in mind here is that with current Internet technology, the QoS of the computation subsystem is mostly under the control of the provider; whereas the QoS of the communication subsystem that links the provider and the consumer depends on the QoS guaranteed by the group of ISPs that compose the path; this path will introduce delays, jitters, packet loss, connection loss and other communication–related disturbances; in processing–intensive application one might consider these disturbances negligible and ignore them; however, there is a large class of applications (for example, auction and stock market services) for which the performance of the communication subsystem is if great relevance.

**Service point of presence** Due to the influence of the communication subsystem, the QoS that a provider can offer to its consumers is not necessarily the same across the whole Internet. Thus, we can imagine that at points $A$ and $B$ the provider will be able to offer a service with $QoS_A$ and $QoS_B$, respectively; whereas at point $C$ the provider offers a service with no guarantee about its QoS, that is, it offers, only best–effort QoS.

We call points of presence of a provider the ISPs to where the provider can deliver its service with a *guaranteed level* QoS. Surprisingly, because of the reasons briefly explained below and discussed at large in [8,25,34], the creation of points of presence is not trivial in the current Internet.

Guaranteed QoS delivered to service consumers results in higher revenues for providers; thus, it is in their interest to deliver a service with guaranteed QoS where its potential consumers are located. Current business tendencies in the ISP market indicate that dominant ISPs are more interested in providing guaranteed communication level QoS only within their own boundaries to offer it as competitive differentiator rather than in collaborating with other ISPs to guarantee QoS over larger areas. Guaranteed QoS over large areas is extremely difficult to provide because it implies collaboration among several autonomous organisations; each of them with their own resources, policies and business goals. Another fact that prevents ISP collaboration is the structure of the relationships between ISPs. Currently, such structure is approximately hierarchical. Between tiers, ISPs are in a customer–provider relationship where the higher–tier (let us say $ISP_A$) is an ISP provider of transport of Internet packets to lower–tier ISPs (let us say $ISP_b$ and $ISP_c$). The higher–tier ISP will often offer its customers SLAs that include clauses about overall packet treatment. For example, $ISP_A$ will offer $ISP_b$ guaranteed level of QoS for the aggregation of packets coming from $ISP_b$ into $ISP_A$ and vice–verse. Unfortunately, higher-tier ISPs normally do not offer SLAs to individual hosts connected to its lower-tier ISPs. The reason for this is that the management overheads are unbearable and the fine grain mechanisms do not work well. Because of this (following our previous example) it is entirely possible for a given host connected to $ISP_b$ to perceive poor performance while $ISP_A$ is still, statistically speaking, meetings its obligations with respect to $ISP_b$. Another fact to take into consideration is that between peer ISPs there are rarely SLAs. For example, it is very uncommon to see SLAs between $ISP_b$ and $ISP_c$ in practice. At the lowest level, ISPs like $ISP_b$ and $ISP_c$ will often offer its customers (individual end users, now) explicit service levels, which typically refer explicitly to delay and loss characteristics at the packet level. These may be statistical (e.g. the 95% of delay will be 100ms between customers of this ISP, or the mean packet loss probability will be no more than 10–5), or they may be bounds (no packet delay will be more than 100ms). SLAs guarantees at the network layer is achieved today typically by network design (provisioning) and is based on extensive measurement and modelling work; this is made possible as network providers now understand the typical source behaviours, and the typical traffic patterns. With the above arguments in mind, as suggested in [25], it seems that currently, the most practicable approach that a provider has to create a point of presence is to share ISPs directly with its service consumers; again, as argued in [25], a provider can increase its number of points of presence by means of multi–homing and collocation.

**Metric collection** Metric collection is at the heart of mechanisms aiming at monitoring SLAs; it is all about gathering statistical information about somebody's activities and involves several issues: (i) Are we using passive (packet sniffing) or active (packet interception, probe with synthetic operations) metric collectors? (ii) From what point or points (provider, service consumer or network in between) are the metrics to be collected? (iii) Who is in charge of collecting the metrics? (iv) What information can be deducted from the collected metrics? With these questions in mind and without paying attention to implementation details, the authors of [25] divide the existing techniques for metric collection into four general categories:

– Service consumer instrumentation
– Provider instrumentation
– Periodic polling with probe clients
– Network packet collection with request-response reconstruction

From a technical perspective (see [25]) each approach has its pros and cons; perhaps a more sensitive choice here is that of the entity in charge of collecting the metrics; this entity has to be a truthworthy, simply because its metrics will become the main source of evidence to solve a dispute between the provider and the service consumer whenever a dispute arises; for this reason, several practical implementation include a trusted third party that collect the metrics, we call this entity, the measurement service.

**An architecture for monitoring non–functional SLAs** A generic architecture for monitoring the level of QoS delivered by a provider to a given service consumer at a given service point of presence $ISP_i$, is suggested in [25]. As shown in Fig. 3, the point of presence is built by means of ISP sharing between the provider and the consumer.

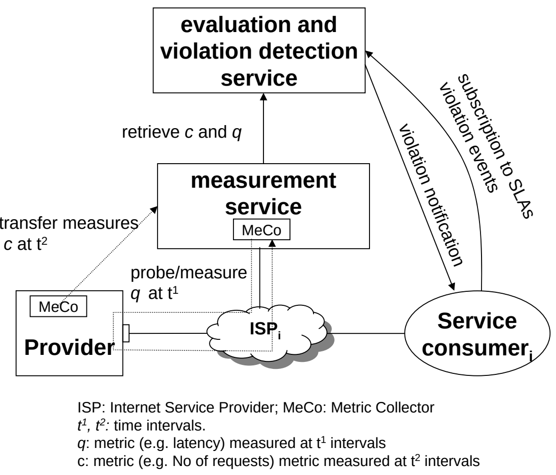

evaluation and violation detection service
retrieve c and q
measurement service
MeCo
transfer measures c at t²
probe/measure q at t¹
MeCo
Provider
ISPᵢ
Service consumerᵢ
violation notification
subscription to SLAs violation events
ISP: Internet Service Provider; MeCo: Metric Collector
t¹, t²: time intervals.
q: metric (e.g. latency) measured at t¹ intervals
c: metric (e.g. No of requests) metric measured at t² intervals

**Fig. 3.** Architecture for unilateral monitoring of non–functional SLAs.

Notice that for simplicity only one point of presence and one service consumer is shown in the figure. However, in a more general scenario, the provider would have one or more points of presence; each of them with an arbitrary number of service consumers which will negotiate possibly different QoS and prices with the provider. Naturally, each provider–service consumer pair will need an instance of the monitoring infrastructure shown in the figure.

To keep the figure and our discussion simple and without loosing generality, we assume for the time being that the provision of the service is unilateral, that is, the service flows only from the provider to the service consumer, as opposite to bilateral provisioning where the two interacting parties provide services to each other; bilateral provisioning is a more general scenario and will be discussed later on in this section. With unilateral service provisioning we need to monitor the observance of only two contractual obligations: (i) the provider's obligations, that dictate that the service must meet certain QoS; and (ii) the service consumer's obligations, which dictate how the service consumer is expected to use the service. The contract is not shown in the figure, however, we assume that it clearly stipulates the QoS at a given point of presence, the metrics that are to be measured and the frequency of metric collection.

The monitoring mechanism shown in the figure relies on two metric collectors (MeCo). One of then belongs to the measurement service and is responsible for monitoring that the provider honours his obligations. Notice that by delegating this job to a trusted third party the service consumer is free from disturbances related to metric collection activities. The second MeCo, that deployed inside the provider monitors that the service consumer meets his obligations.

The specific nature of the metrics to be collected and the collection frequency depends on the application. In the figure we can imagine that the evaluation and detection violation service is retrieving the latest $n$ value of the metric c (number of employees from the service consumer's logged into the provider at a given moment of time), and the latest $k$ values of the metric $q$ (latency of an operation); with these metrics we can compute the latest average latency under the latest average number of users, with an accuracy that depends on the interval ($t_1$ and $t_2$ respectively) with which $q$ and $c$ are measured by the measurement service. Notifications of violations are represented as events. In the figure, these notifications are sent only to the service consumer; however, these notifications can be sent to other parties (for example, to the provider) who express interest by means of subscriptions. The issue about where and how notifications of SLA violations are processed by the service consumer falls out of the interest of this work. However, we can briefly mention that such notifications can be caught by a contract management system that will, after interpreting them, take the necessary actions, such as sending a complaint note or a penalty bill to the offender.

In practice, there are applications where the business partners provide a service to each other, that is, where distinction between the provider and the service consumer is blurred. It is not difficult to see that the architecture presented in the figure can be generalised to work in this more general scenario. A generalisation of this architecture will result in a figure with four MeCo: a MeCo would be deployed inside each party to measure the behaviour of its counterpart seen as a consumer; in addition, two MeCo would be deployed inside the measurement service to probe and measure the performance of each party seen as a provider. We believe that this architecture is general enough and recursive in that it can be placed between any pair of interacting business partners to monitor their interaction. Naturally, it can be placed between $ISP_i$ and the service consumer of the figure to monitor their interaction.

## 3.4 Adaptation

In the adaptation stage, the rubber hits the road. Do the agreements provide the needed information for a system to manage itself automatically? The ability to adapt a system based on the existing agreements is the key behind the autonomic and adaptive infrastructure proposals from companies like HP and IBM, as we will discuss in Section 4. However, in reality, current research and developments rarely consider automated adaptation as the goal for agreements, and it can therefore be expected that the current specification languages need to be modified to appropriately specify all elements needed for automated management. As an example, trade-off decisions require an understanding of rewards and penalties, but many agreement specifications ignore such aspects.

**Violation detection.** An important aspect of self-management is the detection of agreement violations. In violation detection, the following is required:

- detect diversion from acceptable behaviour
- trigger corrective mechanisms
- solve disputes originated from failures to deliver the expected service

Potentially, once can build such a system as a service, which retrieves metrics from the databases of the measurement service, performs computation on them, compares the results of the computation against high or low thresholds and sends notifications of violations to the interested parties when violations of agreements are detected. This service then acts as a trusted third party, as we will discuss in Section 3.5 when we discuss resolution.

**Optimisation Modules.** Additional adaptation can be achieved by executing optimisation algorithms that determine what actions to take to adapt the system. This has been proposed by many authors sometimes based on SLA specifications, e.g., [1,43,39]. However, none of this has been integrated with a multi-party software architecture that involves agreements.

A software architecture that wants to deal with optimisation in general must deal with reliability, scalability, privacy, etc. It is therefore of particular interest to study the implications of distributing optimisation algorithms, both the mathematical foundations and the system considerations. An excellent first step to this can be found in recent work by Nowicki *et. al.* [29].

**Self-Management.** Optimisation modules as mentioned above provide an infrastructure for run-time adaptation of systems, but provides no protocols or message exchange mechanisms to let adaptation 'emerge' in the system. Instead, it provides optimisation outside the system, aiming at a global optimum. In [19] it is argued that that is no true self-management, and does not resolve the scalability issues we face in the management of future large-scale systems.

None of the architectures we discuss in Section 4 is concerned with this issue. That is, none is based on protocols or mechanisms that achieve management through loosely cooperating, let alone emerging behaviour, as discussed in length in [44].

## 3.5 Resolution

The final stage of agreement-based interactions is the termination and after care of the agreement. Obviously, this is particularly important if there are disputes to settle between parties. But also in case of undisputed agreements, dismantling of the monitoring and adaptation software needs to be taken care of, without introducing security and privacy vulnerabilities. To the best of our knowledge, no research about dismantling agreements exist, but it is an important issue.

Related is the issue of introducing changes to existing agreements, realised to be very challenging but not yet much researched. Issues arise about when exactly the new agreement is considered to be agreed upon, since it is not always possible to specify a time instant or unambiguously specify the event that determines the instance the agreement holds. Similar, at what exact moment does the old agreement terminate, and when and how does one dismantle the monitoring and adaptation software? A substantial number of issues arise that require practical solutions.

**Trusted Third Party.** Many authors have mentioned a trusted third party as the appropriate way of determining if agreements have been met, e.g., [24]. The term 'trusted' implies that all parties involved believe monitoring conducted by a trusted third party is done correctly, consequently, they consider outcomes coming from the trusted third party to be authoritative. Based on this idea, one can imagine business models around trusted third parties that monitor, report and discover violations. Companies like Keynote and AlertSite.com have started such businesses, although in a much more static and ad hoc fashion than we imagine in this paper.

If we add the dynamism that we envision, agreements come and go at fast page and in large numbers. When third parties are involved additional protocols are needed to deploy the monitoring software in the third party, exchange data and pass on warnings and other information to the interested party. In the architecture for web services management sketched in [24] the need for protocols has been identified and some candidate protocols have been proposed. Moreover, protocols are also needed in scenarios in which points of presence spread across multiple parties (see Section 3.3).

**Non-repudiation.** Solutions with trusted third parties are a convenient way to avoid dealing with hard technical problems, which one needs to solve if one wants to establish trust between partners. With respect to agreement monitoring, there are at least two sides to trust: can the data that is monitored be trusted to be correct, and is it non-repudiable.

Non-repudiation in broader terms is the concept of ensuring an agreement cannot be denied by any of the parties. For instance, using signatures, none of the parties can claim not to have been involved. Of particular interest is the recent work of Cook *et. al.* [9], which introduces trusted interceptors to achieve non-repudiation. These interceptors insert signatures, and have a protocol for all parties to agree in non-repudiable way on information updates (e.g., monitoring data).

## 4 Existing and Emerging Architectures

Service level agreements are main stay for IP back bone service providers. It is obvious why this is the case, the service they deliver is fully under their control, not relying on any third party. As a consequence, such service providers feel that they are in control of the service they are able to offer. One example is Sprint, which publishes SLAs as well as measured past performance and availability on the open web [41]. The used metrics concern delay, jitter, packet loss and data delivery percentage. For virtual private networks, SLAs are also common place. They especially make sense if the VPN is owned by a single ISP or one that is not Internet connected, since then the ISP has full control over the concerned network. The difficulties in building QoS communication path that expand over several communication providers is discussed in Section 3.3.

A recent study suggests that SLAs are becoming increasingly prevalent in outsourcing deals, some bigger companies reporting to have more than one thousand SLAs closed for their outsourced IT [37]. The same article also mentions that SLA monitoring is still in its infancy, and some times non-existing. Obviously, technologically much progress is still needed, but the service provider model provides a major (and obvious) push for deployment of SLAs.

In what follows we discuss some important existing and emerging software solutions for agreements. They are concerned about agreements in a wider sense, not only to specify service guarantees, but also to keep book on the variety of interactions and to enable automated decision-making.

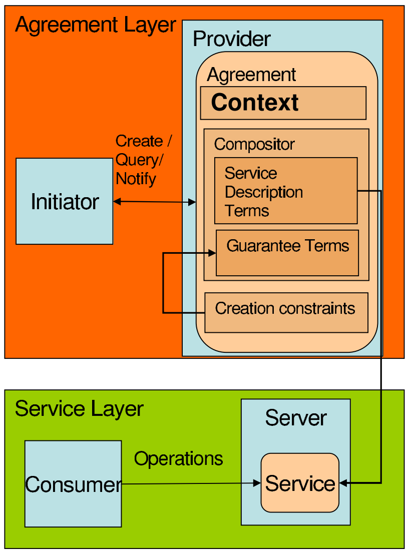


**Fig. 4.** View of WS-Agreement contents and layering.

## 4.1 Global Grid Forum

Web Services are becoming a popular infrastructure for developing distributed systems, particularly in wide-area networks such as the Internet. One of the distinguishing features of Web Services is the large number of activities hosted in standards bodies defining specific functions which can be combined to create an extremely rich environment. Agreements have become a part of this suite via the WS-Agreement specification which has been developed within the Global Grid Forum.

WS-Agreement defines a structure into which an arbitrary set of agreement terms may be placed. The key point is that agreements need not be about any specific type of service (such as a web service), and so can be created and maintained independently of other services. This, in turn, implies that existing service infrastructures need not be changed to introduce agreements. This imposes the layered model shown in Fig. 4. The service layer on the bottom represents the service which is the subject of the agreement. The WS-Agreement model does not change this interaction because the agreement is created, managed, and monitored at a separate logical-layer. One can view an agreement in two ways: as a document or as a service. In the document view, WS-Agreement defines an XML Schema which specifies the components of an agreement. In the service view, an agreement is itself a service which can be monitored and managed in the same way that other services interact with one another. Ultimately, the monitoring infrastructure is likely to have the same architecture as those defined in previous sections.

**WS-Agreement Contents** The agreement document contains the following sub-sections which are also shown in Fig. 4.

– A *Context* contains immutable properties of the agreement as a whole. These include who the provider and consumer of the agreement are, a completion time for the agreement, and references to other agreements which may be related to this one. Related agreements can be used in many ways. One use is to refer to other agreements that are held simultaneously with this agreement to define a larger aggregate agreement. Another is to allow to parties to form a long standing agreement with shorter term, sub-agreements defined for a specific interaction at a particular point in time. WS-Agreement currently does not provide any specifics for how these multi-agreement relationships are formed, specified or monitored.
– *Service Description Terms* describe the service to which the agreement refers. WS-Agreement does not specify the content of them, so they can contain any arbitrary XML schema. In the most simple case, a description term may contain only a reference to an existing service to which the agreement applies. In other cases, these terms could provide detailed specifications of the functional properties for the service to which the agreement will apply. In these cases, it will be common for a new service to be created which conforms to these property definitions.
– *Guarantee Terms* define the non-functional properties of the agreement. Like the service description terms, WS-Agreement does not specify what the contents of the guarantee terms are, but it is expected that they contain enough information that a monitoring system could be configured to enforce the

properties of the agreement. In addition to the non-functional properties, guarantee terms may also contain clauses referred to as "business value" that contain rewards or penalties based on a service provider succeeding or failing in meeting the guarantees.

- *Constraints* are used to narrow the possible values for either the service description or guarantee terms. These are placed into an agreement document called a *template* which can be published to define a providers agreement options. The use of templates is described in more detail below.
- All of the terms are grouped by a *compositor*. The compositor groups the terms, and provides a logical relationship for those terms. The relationships are: "all of," "exactly one of," or "at least one of." Compositors therefore allow for alternative choices within the agreement document. When paired with guarantee terms and business values, this allows a single agreement document to define multiple, acceptable of Service-Levels with corresponding rewards. Compositors can be nested within one another providing an extremely rich structure of alternative and required service description or guarantee terms.

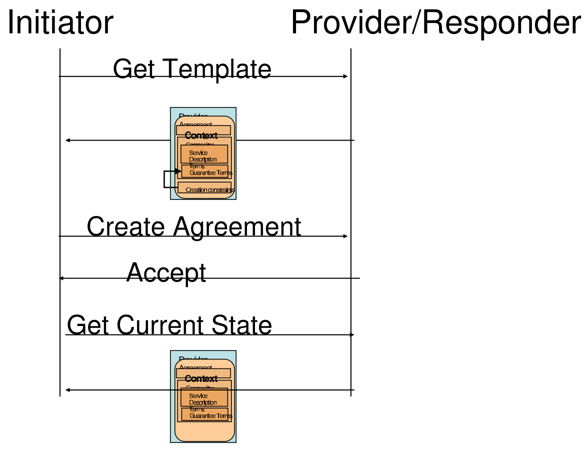


**Fig. 5.** Message exchanges in WS Agreement.

**WS-Agreement workflow** The WS-Agreement specification defines a simple workflow for the advertisement, creation and monitoring of agreements. It is anticipated that these basic functions could be combined to perform more complex interactions such as brokering or negotiation though no specific protocols are presently defined. The basic message exchanges are shown in Fig 5.

The interchange begins when an initiator requests a template document from an agreement provider. This template effectively defines the structure of the agreement it supports by defining the terms and their compositor structure. The template also gives hints to the initiator about acceptable values for those terms via the constraints described previously. In some cases, this template could be generated in response to each request, so the constraints could be used to reflect the current state of the provider.

Upon receiving the template, it is up to the initiator to fill in values for the terms which described the desired agreement. It then sends the proposed agreement document to the provider as a *create* request. The provider can then accept the agreement in which case it returns a positive acknowledgement which also contains a handle to the agreement for use in monitoring. This handle provides a means of interacting with the agreement as a first-class service in a web services environment. If the agreement cannot be reached, an error is returned.

The initiator can use the handle for the agreement to monitor its state. This may be through requests to get an updated version of the agreement document where term values are filled in to represent the current state of the agreement. For example, a guarantee term that specifies the performance level may be filled in with the most recently measured performance observation.

**Status** The first version of the WS-Agreement specification is complete. It has been authored by participants from IBM, Globus, Platform computing and HP, and has entered the public comment period at the GGF at the time of this writing. The GGF working group which developed WS-Agreement is presently working toward defining higher-levels of functionality on top of WS-Agreement such as more complex negotiation protocols, definition of basic guarantee terms which are widely reusable, and possibly profiles for specific interaction models such as brokering or auctioning.

## 4.2 Industrial Developments: HP and IBM

In addition to the above-mentioned network industry, agreements also play an increasingly important role in enterprise IT. Moreover, this area dominates the research and development in software architectures, as we discuss by focussing on two major enterprise IT companies, namely Hewlett-Packard and IBM.

**Hewlett-Packard.** Since the early nineties, Hewlett-Packard has been a (the) leading management software vendor through its OpenView products [28]. OpenView predominantly focuses on monitoring and visualisation of IT operations. Over the years, the software has consistently moved 'up the stack,' expanding the network monitoring functionality to include system and service monitoring. The primary user target for this software are IT administrators and data center operators.

Service level agreements [32] have been part of monitoring software such as OpenView (other examples are Computer Associates, IBM Tivoli and BMC Software for a long time. In a typical setting, an administrator would use SLA thresholds to allow alarms to be triggered when performance deteriorated. If the metrics are chosen wisely (that is, based on service or even business considerations), one can argue that the SLAs assist in monitoring and assuring higher level management goals. These ideas are illustrated well by the NetGather enhancements of OpenView from software vendor ProdexNet [35]. However, in reality the use of SLAs is often restricted to some of the more obvious metrics, such as basic performance and reliability metrics. Moreover, the level of automation to deal with SLA violations is limited, mainly targeting the triggering of alarms.

Such use of SLAs is widespread in all existing management software but is of limited consequences to the exploited software architecture. In particular, powerfully expressive SLA languages are not needed if the metrics differ little across customers, nor are highly effective adaptation algorithms needed if the main objective is to alarm the administrator in time. However, this has dramatically changed with HP's introduction of its adaptive infrastructure software strategy.

HP's adaptive infrastructure envisions that future IT is flexible enough to adapt to any form of change, from newly arriving customers to failing equipment, from changing business environments to sharing resource ownership. To allow for such flexibility, the proposed software architecture has three main characteristics: virtualisation, service-orientation and automation [44]. Virtualisation enables adaptation by substituting hardware-implemented and hardcoded behaviour with software implemented adaptive solutions. Service-orientation based on web services is needed for standards-based interoperability, hiding of heterogeneity, scalability and software reuse. Automation is needed to deliver on the promises of the adaptive infrastructure without having to involve prohibitively many highly-educated technicians.

One can read more about these ideas in [44], or in the many web pages published by HP. HP has chosen to pursue Grid and web service standards as underpinning of the adaptive infrastructure and is leading research and working groups in GGF [2], chairs the organisation itself, and was instrumental in creating the link between web services and grid standards through their introduction of GGF recommendations with web services proposals in the OASIS web services distributed management working group [30]. Recently, HP moved from their monolithic and sizable utility data center product (discontinued in 2004) to new lighter weight adaptive infrastructure offerings based on acquired start-up technologies [31] and research efforts [20].

Within the context of the adaptive enterprise, the main focus with respect to agreements is on the work in the WS Agreement working group, co-chaired by HP. A detailed description can be found in Section 4.1.

**IBM** IBM software strategy is centered around the notion of autonomic computing [18,7]. Autonomic computing suggest that computing systems have capabilities to recover from failures in ways not unlike the human body: locally initiated and emergent (that is, not dictated by a ‘big brother’ style decision-making module). The main driver for autonomic computing is to limit the amount of personnel needed to run the infrastructure: human involvement in IT management is expensive and error prone. Part of the autonomic computing strategy is a focus on on-demand computing and federation through open, standardised web services.

The technology push from IBM is very similar to that from HP. HP’s adaptive infrastructure (discussed in the previous section) is in spirit and in fact similar to IBM’s autonomic computing, and utility computing is largely identical to on-demand computing. Also the business models of the two companies align: from a traditional product focus, the attention is increasingly on services for IT operations, delivered by consultants, very often through ‘outsourcing’ deals. Also from this business perspective the push for automation and on-demand computing models fits nicely, since it makes IT management less expensive

With respect to software architectures, IBM focuses on web services and GGF grid technologies, implemented through open source prototypes or on top of IBM’s Websphere J2EE compliant application server. The work described in Section 4.1 on WS Agreement is heavily influenced by IBM’s involvement. In particular, the proposed specification language WSLA, discussed in Section 3.1, in now being modified to be incorporated in the WS Agreement proposals. In other words, IBM is an important driver to make reality the role of agreements suggested in this paper.

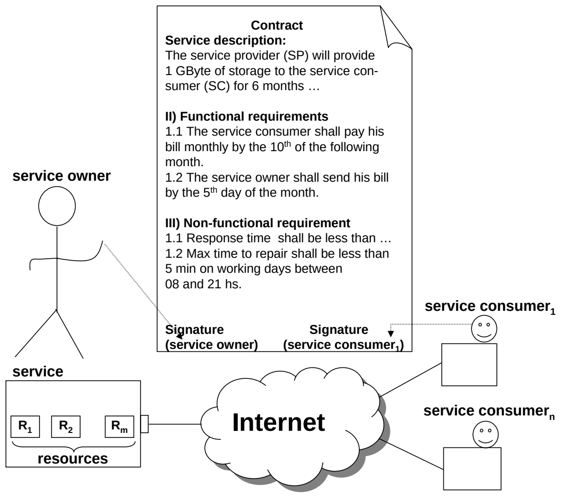
### 4.3 Academic Research: TAPAS

TAPAS (Trusted and QoS–Aware Provision of Application Services) is an on–going research project sponsored by the European Commission. The overall objective of the project is to develop novel methods, tools, algorithms and protocols that support the construction and provisioning of Internet application services with guaranteed levels of QoS. An Internet application service in the TAPAS context is a service of an arbitrary nature (hosting, storage, auctioning, credit rating, stock marketing, etc.) offered to one or more customers simultaneously, over the Internet. TAPAS is relevant to the central topic of our discussion because its developers consider SLAs absolutely crucial to achieve TAPAS overall objective; consequently, one of the aims of the project is to develop QoS enabled middleware capable of meeting SLAs between pairs of interacting parties at different levels.

In TAPAS business scenario a service provider provides its services to several consumers whose access to the service might overlap. The services required by each client are not necessarily the same; neither are the SLAs that they require. As shown in Fig. 6, TAPAS assumes that the service owner is in control of a pool of resources $(R_1, R_2, ..., R_m)$ that he uses to build the services that its consumers require; examples of these resources are cpu, disks, database, servers, etc. Though not explicitly shown in the figure, it is assumed that some of the resources are owned by the service owner, whereas other are hired from other providers though the Internet.

An important assumption in TAPAS is that interacting parties are mutually suspicious and reluctant to engage in business interactions unguarded, because of this, interactions are regulated by legal contracts signed by the interacting parties. Thus, though for the sake of simplicity only one contract is shown in the figure, it should be assumed that the service owner is involved in $n$ contracts, one with each consumer. As shown in the figure, the expectation here is that the legal contract will contain, in addition to conventional contract headers, a list of functional and a list of non–functional requirements, that describe in conventional English prose, the rights and the obligations that the interacting parties are expected to honour. A strong assumption in TAPAS is that SLAs are considered useful if their compliance is monitored and enforced by computer means at run time; this requirement has a significant impact in TAPAS middleware.

The challenge for the service owner is to manage his resources in order to guarantee that the contracted SLAs with his customers are met. TAPAS designer based their solution in the concepts of SLAs specification, monitoring, adaptation, dispute resolution and so on, discussed earlier in this paper. In the following sections we will discuss how these concepts were realised in the TAPAS project.


**Contract**
**Service description:**
The service provider (SP) will provide 1 GByte of storage to the service consumer (SC) for 6 months …

**II) Functional requirements**
1.1 The service consumer shall pay his bill monthly by the 10th of the following month.
1.2 The service owner shall send his bill by the 5th day of the month.

**III) Non-functional requirement**
1.1 Response time shall be less than …
1.2 Max time to repair shall be less than 5 min on working days between 08 and 21 hs.

**Signature (service owner)** **Signature (service consumer$_1$)**

**Fig. 6.** TAPAS business model showing a provider and consumers.

## 4.4 TAPAS architecture

A module architecture of the TAPAS platform is shown in Fig. 7.

The figure shows the main features of the TAPAS architecture. If we ignore the three shaded/patterned entities (these are TAPAS specific components), then

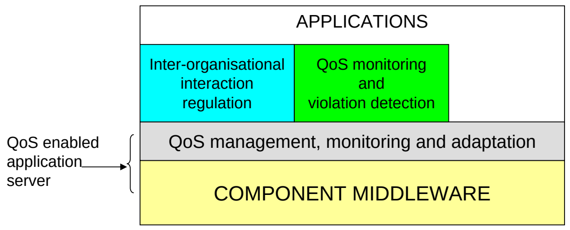

APPLICATIONS
Inter-organisational interaction regulation
QoS monitoring and violation detection
QoS management, monitoring and adaptation
COMPONENT MIDDLEWARE
QoS enabled application server

**Fig. 7.** TAPAS architecture showing its main building components.

we have a fairly 'standard' application hosting environment, that is, an application server constructed using component middleware (e.g., J2EE application server). It is the inclusion of the shaded/patterned entities that makes all the difference.

**Inter–organisation interaction regulation** This module represents the middleware that guarantees that the functional SLAs between two interacting parties are monitored and possibly, enforced, when a violation is detected. For example, for auction applications, it will guarantee that bidders place bid only when bid rounds are declared open. The current version of TAPAS, realises this module as Finite State Machines (FSM) that model the functional SLAs of the original legal contract. Conceptually speaking, this module is located between the two interacting parties to intercept messages exchanged between the interacting parties and prevent the ones that divert from the expected sequence, from reaching the receiving business partner.

**QoS monitoring and violation detection** This module represents the middleware that guarantees that the non–functional SLAs between two interacting parties are monitored and notifications are sent to the interesting parties, when a violation is detected. For example, for auction applications, it will guarantee that response time to a 'PlaceBid' operation is no longer than what it is stipulated in the original legal contract. In the current version of TAPAS, this module is realised as trusted third parties that periodically probe the provider to collect metrics about his performance; in the same order, SLAs are specified in SLAng (see Section3.1) to reduce the level of ambiguity.

**QoS management, monitoring and adaptation** This module represents the middleware to convert conventional application services into QoS enabled ones. For example, this module will contain all the necessary logic to locally monitor the performance that the application service delivers to each customer and the performance of the resource used for building the service; likewise, it will contain algorithms for comparison, tuning, optimisation and adaptation. Current version of TAPAS implements this module as an adaptive clustering mechanism that incorporates QoS awareness into the application service. The current environment consists of a set of Linux computers running instances of the JBoss application server.

It is worth emphasising that, besides the existence of some relationships between the three modules discussed above, their functionality is fairly independent in the sense that a given application does not necessarily need to implement the three modules together; for example, it is conceivable that, to save costs, in an auction application a bidder prefer to exclude non–functional SLAs from his contract; similarly, it is quite possible that a provider in possession of a large number of resources might opt for over provisioning rather than paying for the cost of implementing and running the QoS monitoring and violation detection module.

## 5 Conclusion

This paper surveys the state of the art in technologies for agreements, with an emphasis on the implications for the software architectures. Agreements form the basis for advanced forms of automated management that adapt systems to optimise business objectives, deals with conflict resolutions between partners, and offers non-repudiable evidence for agreement breaches. However, as we point out in the paper, many pieces must come together to achieve such advanced functionality, from agreement specification, automated provision of resources, monitoring and adaptation, to resolution of conflicts. The amount of attention paid to software support for agreements is promising, and we reviewed the advances in standards bodies, industry and academia. However, much research remains to make practical the promise of agreement-based automated system and operation management.

## References

1. G. Alvarez, E. Borowsky, S. Go, T. Romer, R. Becker-Szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, J. Wilkes, "Minerva: an automated resource provisioning tool for large-scale storage systems," ACM Transactions on Computer Systems 19(4):483-518, Nov. 2001.
2. Andrieux et. al., *Web-Services Agreement Specification (WS-Agreement)*, Recommendation track document of the Global Grid Forum, https://forge.gridforum.org/projects/graap-wg/document/WS-AgreementSpecificationDraft.doc.
3. K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalanter, S. Krishnakumar, and D. P. Pazel, J. Pershing,and B. Rochwerger, Oceano–SLA Based Management of a Computing Utility, *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management,* 2001.
4. Algirdas Avizienis, Jean–Claude Laprie, Brian Randell and Carl Landwehr, Basic Concepts and Taxonomy of Dependable and Secure Computing, *IEEE Transactions on Dependable and Secure Computing,* Vol. 1, Nr. 1, pp. 11–33, 2004.

5. C. Bartolini, A. Boulmakou, A. Christodoulou, A. Farrell, M. Salle and D. Trastour, Management by Contract: IT Management driven by Business Objectives, *HP Labs Technical Report, HPL-2004-184*, http://www.hpl.hp.com/techreports/2004/HPL-2004-184.html, 2004.
6. M. Bearden, S. Garg, W.-J. Lee, A. van Moorsel, "User-Centric QoS Policies, or Saying What and How," *11th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2000),* work in progress session, Austin, Texas, USA, Dec. 4-6, 2000.
7. Bloor Research, The Grid Report: The Commercial Implications of the Convergence of Grid Computing, Web Services, and Self-Managing Systems, *Bloor Research North America*, August, 2002.
8. Marjory S. Blumenthal and David D. Clark, Rethinking the Design of the Internet: The End–to–End Arguments vs. the Brave World, *ACM Transactions on Internet Technology*, Vol. 1, Nr. 1, Aug, 2001.
9. Cook, N.O., Robinson, P. and Shrivastava, S.K., Component Middleware to Support Non-repudiable Service Interactions, In *Proc. IEEE Int. Conf. on Dependable Systems and Networks (DSN 2004)*, Florence, Italy, 28 Jun.-1 Jul. 2004
10. Karl Czajkowski, Ian Foster, Carl Kesselman, Volker Sander, Steven Tuecke SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems 8th Workshop on Job Scheduling Strategies for Parallel Processing,Edinburgh Scotland, July 2002
11. A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, A. Youssef, Web Services On Demand: WSLA-Driven Automated Management, IBM SYSTEMS JOURNAL, VOL 43, NO 1, 2004.
12. Yixin Diao, Frank Eskesen, Steven Froehlich, Joseph L. Hellerstein, Alexander Keller, Lisa F. Spainhower, Maheswaran Surendra, "GENERIC ON-LINE DISCOVERY OF QUANTITATIVE MODELS FOR SERVICE LEVEL MANAGEMENT", in *proceedings of IEEE Conference on Integrated Management*, 2003.
13. A. Farrell, D. Trastour, A. Christodoulou, Performance Monitoring of Service Level Agreements for Utility Computing using the Event Calculus, HP Labs Technical Report, HPL-2004-20, 2004.
14. S. Frolund, J. Koistinen, "Quality-of-Service Specification in Distributed Object Systems", *Distributed Systems Engineering Journal,* Vol. 5, No. 4, Dec. 1998.
15. Global Grid Forum Home Page, http://www.ggf.org.
16. P. Goldsack, J. Guijarro, A. Lain, G. Mecheneau, P. Murray and P. Toft, SmartFrog: Configuration and Automatic Ignition of Distributed Applications, *10th OpenView University Association workshop,* Geneva, June 2003.
17. Grid Resource Allocation Agreement Protocol (GRAAP) Working Group: http://forge.gridforum.org/projects/graap-wg.
18. P. Horn, *Autonomic Computing: IBMs Perspective on the State of Information Technology,* IBM, USA, 2002. (Available at http://www.research.ibm.com.)
19. M. Jelasity, A. Montresor and O. Babaoglu, "Grassroot Self-Management: A Modular Approach," in *International Workshop on Self-* Properties in Complex Information Systems,* Bertinoro, Italy, pp. 85–88, May 2004.
20. M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton, F. Gittler, "SoftUDC: A Software-Based Data Center for Utility Computing", *IEEE Computer*, Vol. 37, No. 11, pp.38–47, 2004.
21. Katarzyna Keahey, Takuya Araki, Peter Lane "Agreement-Based Interactions for Experimental Science", in: Proceedings of the EuroPar, 2004.

22. D. D. Lamanna, J. Skene and W. Emmerich, SLAng: A Language for Service Level Agreements, *In Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems* pp. 100–106. IEEE Computer Society Press, 2001.
23. Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, Richard Franck, Web Service Level Agreement (WSLA) Language Specification, Version 1.0, Revision wsla–2003/01/28 *IBM Research Web page*, http://www.research.ibm.com/wsla/documents.html, 2003.
24. V. Machiraju, A. Sahai and A. van Moorsel, Web Services Management Network: An Overlay Network for Federated Service Management, Kluwer, *Proceedings of International Symposium on Integrated Network Management, IM2003,* March 2003.
25. Carlos Molina–Jimenez, Santosh Shrivastava, Jon Crowcroft and Panos Gevros, On the Monitoring of Contractual Service Level Agreements, *The First IEEE International Workshop on Electronic Contracting (WEC), San Diego, 6th Jul* 2004.
26. C. Molina-Jimenez, S. Shrivastava, E. Solaiman, and J. Warne, Run-time Monitoring and Enforcement of Electronic Contracts, *Electronic Commerce Research and Applications,* Elsevier, **3**(2), pp 108-125, 2004.
27. Andrew Moore, James Hall, Christian Kreibich, Euan Harris and Ian Pratt, Architecture of a Network Monitor, *Proc. The Passive and Active Measurement Workshop, La Jolla California, Apr. 6-8*, 2003.
28. N. Muller, *Focus on OpenView: A Guide to Hewlett-Packard's Network and Systems Management Platform*, CBM Books, USA, 1996.
29. T. Nowicki, M. Squillante, C. W. Wu, "Fundamentals of Dynamic Decentralized Optimization in Autonomic Computing Systems," to be published in Lecture Notes in Computer Science, Self-* Properties in Complex Information Systems, Springer Verlag, 2005.
30. OASIS Web Services Distributed Management Technical Committee (WSDM), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm.
31. OpenView Automation Manager, http://managementsoftware.hp.com/solutions/server/demo_0001_transcript.html.
32. OpenView Service Desk SLA management product, http://www.managementsoftware.hp.com/products/sdesk
33. Marcelo Pias and Steve Wilbur, EdgeMeter: Distributed Network metering, *Proc. Int'l IEEE Openarch Conf., short paper session, Anchorage, Alaska, Apr.*, 2001.
34. Panita Pongpaibool and Hyong S. Kim, Providing end–to–end service level agreements across multiple ISP networks, *Computer Networks*, Vol. 46, Issue 1, pp.3–18, Sep, 2004.
35. ProdexNet, independent software vendor, http://www.prodexnet.com.
36. Pruyne, Jim; Machiraju, Vijay Quartermaster: Grid Services for Data Center Resource Reservation HPL Technical Report, HPL-2003-228
37. B. Rosenthal, "A Surprising New Study: SLAs Now Have Teeth," *OutsourcingSLA.com*, http://www.outsourcing-sla.com/surprising.html.
38. A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel, F. Casati, Automated SLA Monitoring for Web Services, 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications, Springer Verlag, pp. 28 41, 2002.
39. C. Santos, X. Zhu, H. Crowder, A Mathematical Optimization Approach for Resource Allocation in Large Scale Data Centers HP Laboratories, Technical Report, HPL-2002-64R1, 2002.
40. James Skene, D. Davide Lamanna and Wolfgang Emmerich, Precise Service Level Agreements, *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, 2004.

41. Sprint back bone SLAs and measured metrics are available at http://www.sprint.com/business/support/serviceLevelAgreements.jsp.
42. Vladimir Tosic, Bernard Pagurek, Kruti Patel, Babak Esfandiari and Wei Ma, *Management applications of the Web Service Offerings Language (WSOL)*, *Information Systems*, In Press, Corrected Proof, Available online 24 December 2004.
43. A. van Moorsel, “The ‘QoS Query Service’ for Improved Quality-of-Service Decision Making in CORBA,” *Symposium on Reliable Distributed Systems,* Lausanne, Switzerland, Oct. 1999.
44. A. van Moorsel, Grid, Management and Self-Management, *The Computer Journal*, to appear, 2005.
45. David Watson, G. Rober Malan and Farnam Jahanian, An extensible probe architecture for network protocol performance measurement, *Software Practice and Experience*, Vol. 34, 2004.
46. A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, S. Waldbusser, *IETF request for Comments RFC 3198,* http://www.ietf.org/rfc/rfc3198.txt, 2001.