

The Role of Agreements in IT Management Software

Carlos Molina-Jimenez¹, Jim Pruyne², and Aad van Moorsel¹

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. Various forms of agreements naturally arise in the service provider model as well as in multi-party computing models such as business-to-business, utility and grid computing. The role of these agreements is twofold: they stipulate obligations and expectations of the involved parties, and they represent the goals to be met by the infrastructure. As a consequence of this latter point, in order to automate run-time adaptation and management of systems and services, agreements should be encoded and integrated in management software platforms. In this paper, we review the state of the art in software support for various forms of agreements, for all stages of their life-cycle. We also review emerging platforms and technologies in standard bodies, industries and academia.

1 Introduction

We will argue and illustrate in this paper that distributed computing infrastructures must incorporate *agreements* as first-class software building blocks 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. Additionally, these agreements are means to set the objectives for automated decision-making in system adaptation and management. It can therefore be useful for an IT operator to formulate objectives in the form of agreements even if no other parties are exposed to this information.

Modern-day and emerging computing infrastructures are increasingly flexible in their support of computational and business models, as witnessed by the developments of adaptive and on-demand computing solutions advocated by HP, IBM, Oracle, SUN and others. These solutions typically envision a service provider model for various aspects of computing, such as CPU use, network use, application hosting, etc. As software platform, such solutions are often tied to the grid [20], which supports resource sharing across multiple parties using open software. This software virtualises resources and applications, thus shielding the customer from the complexities of the underlying infrastructure and providing the operator with tools to adapt the system gracefully at run-time.

To enable the mentioned multi-party computing models the supporting software infrastructure should make use and keep track of the agreements established between parties. Therefore, it should embody these agreements in software. Such *run-time agreements* can come in various shapes or forms (at times we will use the adjective ‘run-time’ to stress that we discuss software embodiments of agreements, used at run-time to manage the execution of services). For instance, a run-time agreement can be defined by an operator to represent aspects of a hard-copy contract signed between provider and customers. Alternatively, the run-time agreement represents agreed-upon service levels automatically negotiated by software agents of the provider and customer. Irrespective, the information in the agreement can be used throughout the platform as needed, e.g., to adapt the system to meet service levels while optimising profits. Agreements thus naturally fit the service provider model, but also provide the necessary information to allow for automated decision-making by management software and self-managing components and services.

Since this article has been prepared for the series of books on architectures for dependable systems, it is opportune to address the relation of agreements with both dependability and architectures. The notion of architecture used in this paper relates to the structure of software platforms (middleware). These architectures will be heavily influenced by the emergence of the service provider computing model (including utility or on-demand computing), and by an increased pressure to automate system operation and hence save operational cost. We foresee a prominent role for agreements, which will be integrated in such architectures and will be represented as objects, services or other software components. Once these are in place, they define the objectives to be used in the algorithms that adapt systems and services.

With respect to dependability, run-time agreements play a double role, as indicated above. On the one hand, agreements must be available in system operations software to determine how to adapt the system, also in response to failures. This entails further automation of the processes traditionally involved in fault management. On the other hand, agreements are a way of providing trust in the system, by allowing customers to express their interests, and by providing them with information about whether the agreements are met (possibly through a trusted third party).

There are many open issues in the technologies required to support run-time agreements. The emphasis in this paper is on a survey of existing and ongoing work related to software architectures, with pointers to remaining research issues. The survey is extensive, but arguably not exhaustive because of the vastness of the area we want to cover. In Section 3 we discuss 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 representative solutions proposed by standards bodies (WS-Agreement in the Global Grid Forum), industries (Hewlett Packard and IBM enterprise IT) and academia (TAPAS, an EU research project). To set the stage, we first discuss terminology used in this paper and in the literature.

2 Definition and Terminology

In this paper we research the software embodiment of contracts or other types of agreement or objectives. As mentioned above, at times we will use the term *run-time agreement* to stress we are concerned about utilising these agreements during service execution. We define run-time agreement as follows.

A *run-time agreement* is a machine interpretable representation of agreed-upon service characteristics or objectives, established between two (logical) parties. These run-time agreements are used as the goals that drive some form of automation. For this purpose, multiple run-time agreements may be combined to establish the overall objective.

Software embodiments of agreements have been proposed and implicitly used in various places, predominantly in the area of policy-based management. For instance, in the Gallifrey project at Bell Labs [7], goals bound to policy rules form first class objects in the software architecture. However, most policy work (see [35] for a survey in network management) focuses on the rules instead of goals [8].

With the emergence of the service provider model, the distinction between goals and rules becomes much more important. It does not make sense for a customer of a utility computing data centre to set policy rules that manage the system. Instead, this is the responsibility of the service provider, and as a consequence policy rules and agreed-upon objectives must be decoupled. It also is not appropriate to assume that written documents can be used to define the service level agreements, and that operators manually adapt policies based on these documents. Utility computing or other service provider operations can be expected to be much too dynamic to allow such slow processes.

The consequence of the above is that open software and standards are emerging that allow run-time agreements to be exchanged between parties. The utility system is then managed and adapted based on these run-time agreements. Eventually, we expect system management to be so intimately tied to run-time agreement that operators and administrators will introduce additional agreements in the system, to modify the goals driving the management system. In this process, system operators move more and more from defining and refining policy rules to defining and refining goals (as advocated in [8], among others).

It may be good to compare our definition of agreement with existing related terms and definitions, for instance the definition of service level agreement (SLA) in IETF RFC 3198 [61]. There, 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 Quality of Service or other attributes of the service.” A service level objective is “a set of parameters and their value.” This is a very natural definition, although the explicit requirement that parties negotiate is not always valid in our setting.

More important is the fact that in the current paper, we consider machine-interpretable agreements, and focus on run-time software components representing agreements. This was not the objective behind RFC 3198. Note furthermore

that the RFC 3198 definition also points to QoS, which is a typical connotation when SLAs are concerned. In this paper we do not restrict the contents of run-time agreements to QoS. For a good understanding, it is important to realise that (service level) agreements are not synonymous to QoS. QoS is about metrics (performance, up-time, throughput), which may or may not be used in an agreement or SLA to specify expected values, associated penalties, etc.

It is also of interest to clarify the relationship between policies and agreements, since both are often used in system management. The term policy usually refers to “a plan of actions” [31], while agreements are used to denote a set of goals. Ideally, the goals represented in run-time agreements should determine (or be used to parameterise) the actions suggested by the policies (see [8] for a further discussion of this topic). However, it is important to note that agreements can also be used for adaptation of a system by other means than policies, e.g., using games, auctions or run-time mathematical optimisation algorithms (see Section 3.4).

3 Software Infrastructure Requirements

We discuss requirements for software support of agreements in five groups, roughly corresponding to stages in the life-cycle of an agreement.

1. **Specification.** Languages and formalisms for expressing the agreement, choice of metrics, negotiation, run-time embodiment.
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 detection, third-party involvement, data exchange protocols.
4. **Adaptation.** Decision-making about new requests and adapting resource allocations, automated response to changes, business-driven adaptation, alarm handling, self-management and autonomic management.
5. **Resolution.** Auditing and non-repudiation, validity of and changes in agreements, conflict resolution.

Within each item, we touch on a diverse range of issues, from dependability requirements and automation needs to standardisation and third-party solutions. Table 1 summarizes our findings.

3.1 Specification

There is a large body of literature available on the specification of service level agreements, but far less on the formatting of a run-time agreement as a software building block, for instance through the definition of interfaces. The latter is of particular interest to us, since it determines the information the software platform makes available for automation. The obvious exception is WS-Agreement, which we discuss in Section 4.1.

Table 1. State of the art and open issues

	Status	Open or Underdeveloped Issues
Specification	-threshold SLAs in commercial management software [45,48] -QoS languages [2,17,19,21][28,34,53,55] -WSLA [15,33] adopted in WS-Agreement [3] -WSOL extension to WSDL [57]	-practical mix of expressiveness and simplicity -tool support to define agreements [34] -penalties and rewards [16,62] -business metrics and business-driven management [9,11,36,62] -(domain-specific) ontologies -(automated) negotiation [27,30]
Provision	-job and workload scheduling [41] -resource provision [2,5,22,49] -monitoring provision [15,53]	-mapping (business) metrics on resource needs [16,36] -inclusion of QoS concerns [2,5,18] -domain or product-specific deployment templates [22]
Monitoring	-automatic monitor ignition [15,39,53] -third-party monitoring [46] -threshold alarms [45]	-measurement exchange protocols [37] -business-driven monitoring [9,11] -performance assessment [41] -scaling to very large scale networks -techniques to enhance trust
Adaptation	-(on-line) optimisation algorithms [42,54] -dynamic resource allocation [5,13,41] -domain-specific research prototypes [12,50,51]	-guaranteed end-to-end QoS [47,51] -signalling protocols [18,37] -aggregated objective from multiple agreements -secure sharing of agreements within and across platforms -service definition extension with adaptations [24,32] -self-management, emerging behaviour [6,23,26,59]
Resolution	-customer credits [46] -non-repudiation with trusted third party [14]	-changing and terminating agreements -resolution protocols -enhanced trust solutions

We envision many run-time agreements to be present at any point in time, representing agreements with different customers and possibly additional provider objectives. In software terms, each run-time agreement may be represented by an object or a service, or they will be aggregated into one software component to make them simpler to track. Irrespective, to be useful for automation run-time agreements must contain enough information to facilitate decision-making: sufficiently detailed statements about expected functionality, exact enough quality-of-service parameters and values, and reward and penalty information for met and missed objectives. The latter is particularly important: without rewards and penalties a decision-making unit can not make trade-off decisions in times of scarcity. A serious shortcoming in many current agreement

specification efforts (WS-Agreement is an exception) is the lack of attention paid to rewards and penalties, possibly expressed in monetary values [9,11,36]. This ultimately will result in suboptimal decisions during the adaptation phase (discussed below).

In the web services area service descriptions and workflow-style service definitions are commonplace through the Web Service Description Language (WSDL), Business Process Execution Language, and similar efforts [1]. For every client of the web service, the web service run-time system maintains information to manage the interactions with the client. Arguably, this corresponds to maintaining a run-time agreement, as we defined in Section 2. Two approaches can be thought of to populate such run-time agreements with sufficient information for system management. First, one can leave it to the operator to provide the management system with additional information about priorities across customers, QoS guarantees for customers, rewards and penalties for meeting and missing objectives, etc. This will require operator tools to facilitate inputting such information. It is not impossible that commercial software vendors are thinking in this direction, although we are not aware of other efforts than the WS-Agreement related research demo Cremona [34].

Alternatively, one can extend the service definition with additional information, as is done in Web Service Offering Language (WSOL) [57]. WSOL is fully compatible with the WSDL description language and extends WSDL with capabilities relevant to service offerings. 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 WSOL envisions that a web service offers a service S as a set of classes c_1, c_2, \dots, c_m , where c_i and c_j offer the same functionality but differ in terms of constraint parameters.

Other related work has been focused predominantly on unambiguous representation of QoS metrics [21,33,53,55], sometimes through the application of formal methods [19,39]. For a recent overview of QoS definition technologies we refer to [17,28]. Of course, no matter which formal approach is used, the problem of semantic meaning of objectives will remain, and it may therefore be that intricate or complete languages will lose out in practice to descriptions with succinct but sufficient expressiveness. Furthermore, it may be needed to invest in domain-specific ontologies for agreements, scoped such that one can achieve a reasonably precise and widely-accepted understanding of terms in the domain of concern.

3.2 Provision

In this section we consider the first phase in run-time use of agreements, 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.

In terms of the software architecture we can identify a range of possible approaches to standardisation of provision software. On one end, the 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 typical usage scenarios for the grid job submission description language [4], or utility computing definitions in the SmartFrog language [22]. Alternatively, the agreement is represented by a first-class software object (or service) with standardised interfaces. This leaves a more powerful interaction paradigm and is the idea behind WS-Agreement, which we discuss in detail in Section 4.1. Of course, independent of the chosen approach, the provision system needs to keep track of the various agreements in its software.

Provision systems such as SmartFrog predominantly focus on core tasks behind service provision, such as loading, initialising and starting software components, in prescribed order. Many challenges exist when one needs to provide for additional agreement parameters such as QoS, although research attempts exist in various domains [2,5,18,50]. When we review the technologies of HP and IBM in Section 4, we see these are driven by utility or on-demand computing opportunities [29]. The key enabler of utility computing is software for automated provision, such as SmartFrog [22] and Oceano [5]. Enrichment of such software to be governed by agreements is needed to deliver on the promise of fully automated management in multi-party setting.

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 barriers in using monitoring software is 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 agreements which monitoring software 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 in [37,53] in the context of web services.

3.3 Monitoring

When run-time agreements are used to guide system or service management, the issue of monitoring can be dealt with in straightforward manner. An abundance of monitoring tools is available to collect metrics from almost all elements in the infrastructure, both commercial, e.g., [40], and Open Source [60]. As we indicated above, the challenge lies in automatically igniting the correct monitoring software based on the specification of the present run-time agreements [37,53].

When multiple parties are interested in the monitored data, matters immediately become much more intricate. Each metric in the agreement will be monitored at one of the two parties, and the results will be reported to the other. The question then is why the second party should believe the reported results. All present solutions to this problem rely on a trusted third party [14]. At some level, a trusted third party is unavoidable, but trust-enhancing techniques for non-repudiation, privacy and authenticity remain of prime interest.

Especially when relying on trusted third parties it is desirable to allow agreements to be dynamically initiated, without cumbersome set-up procedures involving the third party. This implies that protocols must be in place to exchange the needed set-up information and credentials, disseminate data to the right parties, and issue actions to adapt the monitoring software to the new agreements. A possible solution to the problem is the introduction of a signalling network overlay, such as the ones proposed in the context of differentiated IP services [18] and web services [37].

One step beyond monitoring is the inference of agreement violations using the measured data [45]. Based on this inference, corrective actions can be triggered and remaining disputes can be dealt with. Potentially, one can build such a full-blown agreement violation management 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.

3.4 Adaptation

To be useful at the adaptation stage the agreements must provide management software with the necessary information for system adaptation. The ability to adapt a system based on the existing agreements is the key behind the autonomic and adaptive infrastructure proposals from IBM and HP, 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.

We envision automated management at the service provider site to be effectively hidden from the customer. However, one could take a different perspective and assume that the service or workflow description of the service includes a specification of adaptations, and are thus visible to and possibly parameterisable by the customer. Technologies based on such an approach have been proposed in [24,32].

There is a rich body of literature on adaptive middleware to guarantee QoS properties (e.g., [12,50], a survey can be found in [51]). For various application areas interesting results are available from these projects. The current paper takes a more generic view at adaptive systems, targeting open software platforms for general-purpose computing.

Mathematical optimisation algorithms can be used on-line and at run-time to optimise the actions taken by the adaptive system. The use of such algorithms has been considered by many authors, sometimes based on SLA specifications, e.g., [2,7,13,54,58]. However, none of this work has been concerned with how to integrate such techniques in an open standardised software architecture. Such a software architecture for mathematical decision making algorithms would have

to deal with reliability, scalability, privacy and many other issues. It is therefore of particular interest to study the implications of distributing the optimisation algorithms. An interesting first step to approach the mathematical and system issues can be found in [42].

Mathematical optimisation modules as mentioned above provide an infrastructure for run-time adaptation of systems, but provide 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 [26] it is argued that this is not true self-management, and does not resolve the scalability issues we will face in the management of truly large-scale systems. Indeed, as discussed at length in [59], none of the architectures discussed in Section 4 achieves management based on emerging behaviour. If this lack of self-management indeed turns out to hurt scalability, research in biologically or socially inspired emerging behaviour in computing systems may be of interest, e.g., [6].

3.5 Resolution

The final stage of agreement-based management 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 exists, 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 amount of research will be needed to find practical solutions for these problems.

In practice, a trusted third party is often considered the appropriate approach to determine if agreements have been met [46]. 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 (compare this with similar monitoring products and services for traditional web sites from companies like Keynote and AlertSite.com).

A property one would like to establish when information is being exchanged between customers and providers is that of non-repudiation. Non-repudiation ensures that the party distributing the information can not successfully deny at a later stage knowing about this information. Of particular interest is the recent work of Cook *et. al.* [14], 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).

If future systems exhibit the amount of dynamism we envision in this paper, agreements come and go at fast pace and in large numbers. When third parties are involved additional protocols are needed to deploy the monitoring software from the third party, exchange data and pass on warnings and other information to the interested party [37]. Hence, as for all other stages, also for the resolution stage one can imagine protocols to be developed that take care of issues that involve multiple parties. For instance, these protocols could report violations and distribute credits to customers if agreements are not met. Such scenarios are speculative, but open up interesting research opportunities.

4 Existing and Emerging Architectures

Service level agreements are already main stay for IP back bone service providers. For example, Sprint openly publishes SLAs as well as measured past performance and availability on the world wide web [56], as do other backbone providers. The used metrics concern delay, jitter, packet loss and data delivery percentage. Backbone providers are willing and able to guarantee, measure and expose service levels because the service they deliver is under their control, not requiring a third party for networking (but of course relying on power supply, hardware and software reliability, etc.). Also when acquiring a virtual private network (VPN) an SLA is commonly agreed upon between customer and provider [46]. GTE Networks reportedly uses a third party for monitoring, and provides ‘credits’ to customers when SLAs are not met [46]. All contracts signed between customers and providers contain caveats related to dependency on other parties in the service delivery. A staggering amount of metrics and disclaimers can for instance be found in JANET’s SLAs for the UK’s higher education network [25].

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 [52]. Remarkable enough, SLA monitoring is still in its infancy and some times non-existing [52], but that apparently does not negate the value of agreeing on an SLA. In general, one would expect that the service provider model will provide a further push for deployment of SLAs.

All the above has had relatively little influence on the software architectures on which services are build. This must change when the services become more complex and the service usage becomes more dynamic. If customers come and go continuously (for instance when using computing equipment for scientific computations) or when the application is extremely intricate (for instance when customers are load balanced across application servers), integration of agreements in the software architecture becomes necessary. In what follows we discuss some important existing and emerging software solutions that aim at that vision.

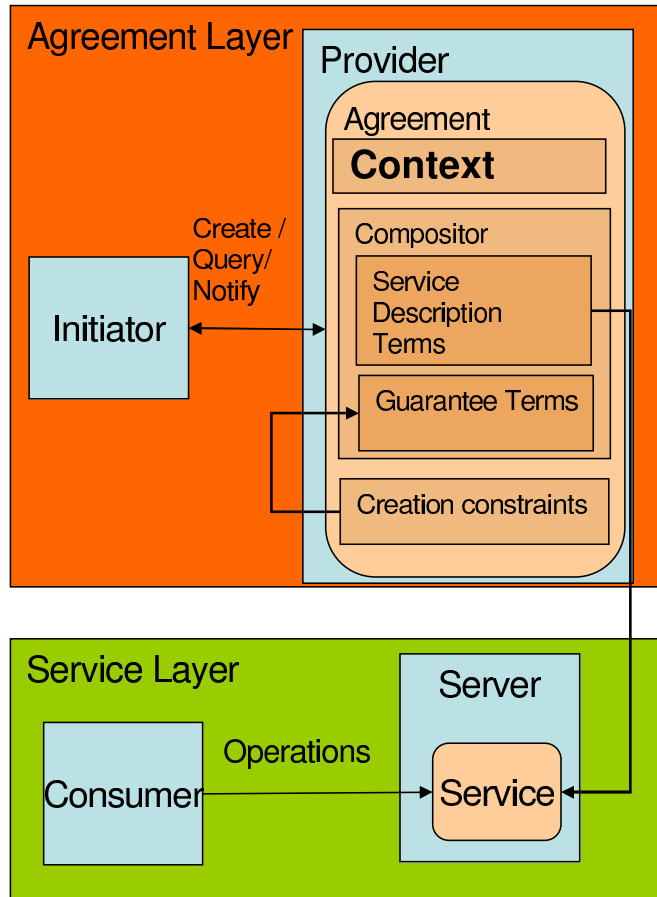


Fig. 1. View of WS-Agreement contents and layering

4.1 Global Grid Forum

Web services are becoming a popular infrastructure for developing distributed systems, particularly in Internet and Intranet setting. 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 [3], which has been developed within the Global Grid Forum (GGF).

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. 1. 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.

WS-Agreement Contents. The agreement document contains the following sub-sections which are also shown in Fig. 1.

- 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 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 simplest 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

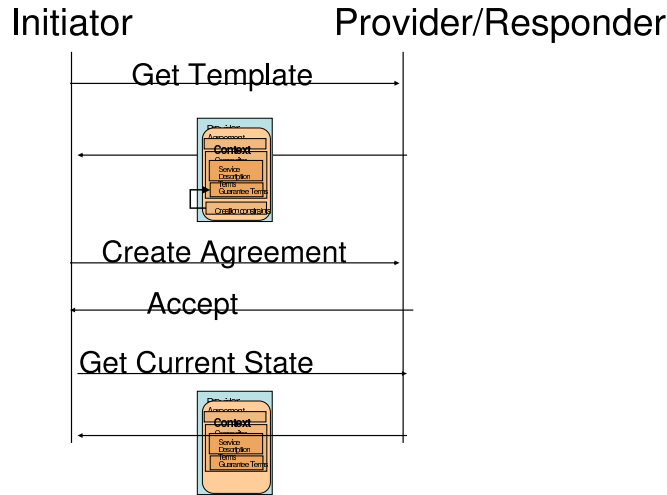


Fig. 2. Message exchanges in WS-Agreement

with guarantee terms and business values, this allows a single agreement document to define multiple, acceptable 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.

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 2.

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 describe 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 Globus, HP, IBM and Platform Computing, and has entered the public comment period at the Global Grid Forum 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

As we mentioned above, network providers are using SLAs extensively, but this has had limited influence on software architectures. Instead, the recent flurry of research and development related to agreements and software architectures is driven by enterprise IT industry. We discuss this by reviewing the technology strategy of two major enterprise IT companies, namely Hewlett-Packard and IBM.

Hewlett-Packard. Since the early nineties, Hewlett-Packard has been a leading management software vendor through its OpenView products [40]. 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 [45] have always been part of monitoring software such as OpenView and that from its competitors Computer Associates, IBM Tivoli, BMC Software, etc. In a typical setting, an administrator uses SLA thresholds to trigger alarms when performance deteriorates. If the metrics are chosen wisely (that is, based on service or even business considerations [9]), the SLAs assist in assuring higher level management goals. These ideas are illustrated well by the NetGather enhancements of OpenView from software vendor ProdexNet [48]. 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 targeted to triggering alarms.

Such use of SLAs is widespread in all existing management software but is of limited consequences to the exploited software architecture. 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. For HP, this has dramatically changed with the 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 demands to sharing resource ownership. To allow for such flexibility, the proposed software architecture has three main characteristics: virtualisation, service-orientation and automation [59]. Virtualisation enables adaptation by substituting hardware-implemented and hard coded 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 vision without requiring prohibitively many highly-educated technicians.

One can read more about these ideas in [9,59], 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 [3], chairs the organisation itself, and was instrumental in creating the link between web services and grid standards through their introduction of GGF recommendations in the OASIS web services distributed management working group [43]. 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 [44] and research efforts [29].

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, see Section 4.1.

IBM. IBM's technology strategy is centred on the notion of autonomic computing [10,23]. Autonomic computing suggests that computing systems exhibit capabilities to recover from failures in ways not unlike the human body's immune system: locally initiated and emergent (that is, not dictated by a 'big brother' style decision maker). The main driver for autonomic computing is to limit the amount of personnel needed to run the infrastructure, since human involvement in IT management is expensive and often a source for failures. 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 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 fits nicely, since it makes IT management less expensive.

IBM product technologies around self-healing storage equipment and web server load balancing have made autonomic computing tangible. Research prototypes such as Oceano [5] have provided an emphasis on software for highly adaptive systems. 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 web service level agreement specification language WSLA [15,33] has been modified to be incorporated in the WS-Agreement proposals.

4.3 Academic Research: TAPAS

We discuss one academic project we have been involved in since it contributes some interesting technologies to agreement-driven service management [14,38,39,55]. The project is called TAPAS, which stands for Trusted and QoS-Aware Provision of Application Services, which has as one aim to develop QoS enabled middleware capable of meeting SLAs between pairs of interacting parties. It is representative for a range of adaptive middlewares, such as those surveyed in [51], but is of particular interest because of its focus on the service provider model.

In a typical TAPAS 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 and neither are the SLAs that they

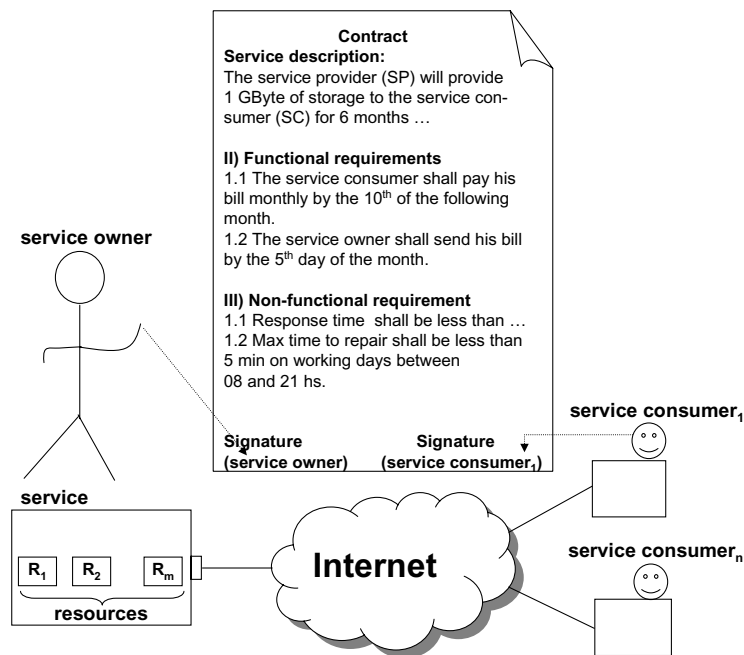


Fig. 3. TAPAS scenario showing provider, consumers and contract

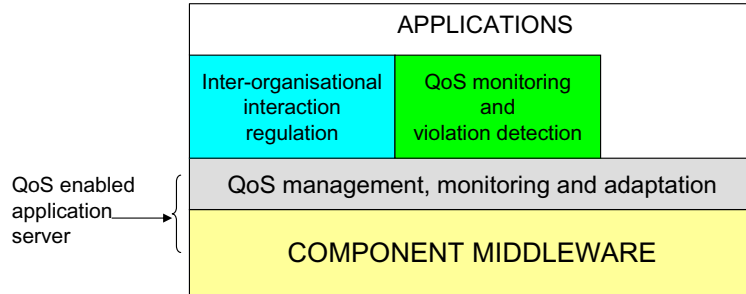


Fig. 4. TAPAS architecture showing its main building components

require. As shown in Fig. 3, TAPAS assumes that the service owner is in control of a pool of resources (R_1, R_2, \dots, R_m) used 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 provider, whereas others are hired from other providers over the Internet.

In TAPAS 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. For the sake of simplicity only one contract is shown in Fig 3, but it should be assumed that the service provider is involved in multiple contracts, one with each consumer. As shown in the figure, legal contracts 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. 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 provider is to manage his resources in order to guarantee that the contracted SLAs with his customers are met.

A module architecture of the TAPAS platform is shown in Fig. 4. Without the three shaded entities a fairly ‘standard’ application hosting environment would emerge, that is, an application server constructed using component middleware (e.g., J2EE application server). It is the inclusion of the TAPAS specific entities that makes all the difference, as we describe.

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 guarantees that bidders place bids only when bid rounds are declared open. The current version of TAPAS realises this module as finite state machines that model the functional SLAs of the original legal contract [39]. 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 interested parties when a violation is detected [38]. For example, for auction applications, it guarantees that the time a response to a ‘PlaceBid’ operation takes is less or equal to what is stipulated in the original legal contract. In the current version of TAPAS, this module is realised as (logical) trusted third parties, which periodically probe the provider to collect metrics about its performance [14]. TAPAS also proposes the SLAng language for precise specification of SLAs [55] 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 contains 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 contains algorithms for comparison, tuning, optimisation and adaptation. The 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 the three modules discussed above are 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 prefers 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 IT management software platforms. Agreements form the basis for a variety of advanced forms of automated management, enabling functionalities such as adapting systems to optimise business objectives, resolving conflicts resulting from agreement violations, and gathering non-repudiable evidence about agreement breaches. As we point out in the paper, many pieces must come together to achieve such advanced functionality, requiring technological advances in five categories: agreement specification, automated provision of resources, monitoring, adaptation, and resolution of conflicts. We

reviewed recent advances in all these areas, summarizing both state of the art and open problems. We also discussed in more detail the technologies emerging from industry (HP and IBM), and the software platform support being developed in standards bodies (WS-Agreement) and academia (TAPAS). The recent flurry of attention for software support for agreements is encouraging. However, as can be seen in the overview table (Table 1), many challenges remain to be addressed on the road to comprehensive automated agreement-based system and service management solutions.

Acknowledgement

The work in this paper has been funded in part by the UK Engineering and Physical Sciences Research Council, under e-Science Grant No. GR/S63199/01 (Trusted Coordination in Dynamic Virtual Organisations), and by the European Union under Projects IST-2001-34069: TAPAS (Trusted and QoS-Aware Provision of Application Services), and IST-2001-37126: ADAPT (Middleware Technologies for Adaptive and Composable Distributed Components).

References

1. G. Alonso, F. Casati, H. Kuno and V. Machiraju, *Web Services: Concepts, Techniques and Applications*, Springer, 2004.
2. 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*, ACM, vol. 19, no. 4, pp. 483–518, 2001.
3. A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke and M. Xu, *Web-Services Agreement Specification (WS-Agreement)*, Recommendation track document of the Global Grid Forum, 2004.
4. A. Anjomshoaa, F. Brisard, A. Ly, S. McGough, D. Pulsipher and A. Savva, *Job Submission Description Language (JSDL)*, pre-release draft for the Global Grid Forum, 2005.
5. K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalanter, S. Krishnakumar, D. Pazel, J. Pershing and B. Rochwerger, "Oceano-SLA Based Management of a Computing Utility," *IFIP/IEEE International Symposium on Integrated Network Management*, IEEE Computer Society Press, pp. 855–868, 2001.
6. O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel and M. van Steen (Eds.), *Self-Star Properties in Complex Information Systems*, Springer, LNCS vol. 3460, 2005.
7. M. Bearden, S. Garg, A. van Moorsel and W. Lee, "Gallifrey: A Component-Based Framework for Building Policy-Based Management Applications," *Bell Labs Research, Technical Memorandum BL011356-000120-01*, Lucent Technologies, Bell Laboratories, 2000.
8. M. Bearden, S. Garg, W. Lee and A. van Moorsel, "User-Centric QoS Policies, or Saying What and How," *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, work-in-progress report, 2000.

9. C. Bartolini, A. Boulmakoul, 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*, HP Laboratories, 2004.
10. Bloor Research, *The Grid Report: The Commercial Implications of the Convergence of Grid Computing, Web Services, and Self-Managing Systems*, Bloor Research North America, 2002.
11. M. Buco, R. Chang, L. Luan, C. Ward, J. Wolf and P. Yu "Utility Computing SLA Management Based Upon Business Objectives," *IBM Systems Journal*, IBM, vol. 43, no. 1, 2004.
12. W. Chen, M. Hiltunen and R. Schlichting, "Constructing Adaptive Software in Distributed Systems," *International Conference on Distributed Computing Systems*, IEEE Computer Society Press, pp. 635–643, 2001.
13. L. Cherkasova, W. Tang and S. Singhal, "An SLA-Oriented Capacity Planning Tool for Streaming Media Services," *IEEE Conference on Dependable Systems and Networks*, IEEE Computer Society, pp. 743–752, 2004.
14. N. Cook, P. Robinson and S. Shrivastava, "Component Middleware to Support Non-Repudiable Service Interactions," *IEEE Conference on Dependable Systems and Networks*, IEEE Computer Society, pp. 605–614, 2004.
15. A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer and A. Youssef, "Web Services On Demand: WSLA-Driven Automated Management," *IBM Systems Journal*, vol. 43, no. 1, pp. 136–158, 2004.
16. Y. Diao, F. Eskesen, S. Froehlich, J. Hellerstein, A. Keller, L. Spainhower and M. Surendra, "Generic On-Line Discovery of Quantitative Models for Service Level Management," *IFIP International Symposium on Integrated Network Management*, G. Goldszmidt and J. Schönwälder (Eds.), Kluwer, pp. 157–170, 2003.
17. G. Dobson, "Quality-of-Service in Service-Oriented Architectures," *Dependability Infrastructure for Grid Services Project*, <http://digs.sourceforge.net/papers/qos.html>, 2004.
18. G. Fankhauser and D. Schweikert "Service Level Agreement Trading," ETH Computer Engineering and Networks Lab Technical Reports, ETH Zurich, no. 59, 1999.
19. A. Farrell, D. Trastour and A. Christodoulou, "Performance Monitoring of Service Level Agreements for Utility Computing using the Event Calculus," *HP Labs Technical Report, HPL-2004-20*, HP Laboratories, 2004.
20. I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications*, SAGE Publications, vol. 15, no. 3, pp. 200–222, 2001.
21. S. Frølund and J. Koistinen, "Quality-of-Service Specification in Distributed Object Systems", *Distributed Systems Engineering Journal*, Institute of Physics, vol. 5, no. 4, 1998.
22. P. Goldsack, J. Guijarro, A. Lain, G. Mecheneau, P. Murray and P. Toft, "Smart-Frog: Configuration and Automatic Ignition of Distributed Applications," *10th OpenView University Association workshop*, June 2003.
23. P. Horn, *Autonomic Computing: IBMs Perspective on the State of Information Technology*, IBM, 2002.
24. S. Hwang and C. Kesselman, "Grid Workflow: A Flexible Failure Handling Framework for the Grid," *IEEE International Symposium on High Performance Distributed Computing*, IEEE Computer Society Press, pp. 126–137, 2003.
25. JANET/UKERNA, *JANET/UKERNA Service Level Agreements*, <http://www.ja.net/documents/sla.html>.

26. M. Jelasity, A. Montresor and O. Babaoglu, "Grassroot Self-Management: A Modular Approach," in *Workshop on Self-* Properties in Complex Information Systems*, University of Bologna, O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel and M. van Steen (Eds.), pp. 85–88, 2004.
27. N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra and M. Wooldridge, "Automated Negotiation: Prospects, Methods and Challenges," *Group Decision and Negotiation*, Springer, vol. 10, no. 2, pp. 199–215, 2001.
28. J. Jin and K. Nahrstedt, "QoS Specification Languages for Distributed Multimedia Applications: A Survey and Taxonomy," *IEEE Multimedia Magazine*, vol. 11, no. 3, pp. 74–87, 2004.
29. M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton and F. Gittler, "SoftUDC: A Software-Based Data Center for Utility Computing", *IEEE Computer*, vol. 37, no. 11, pp. 38–47, 2004.
30. K. Keahey, T. Araki, P. Lane "Agreement-Based Interactions for Experimental Science", *IFIP/ACM/IEEE Euro-Par*, M. Danelutto, M. Vanneschi and D. Laforenza (Eds.), Springer, LNCS vol. 3149, pp. 399–408, 2004.
31. J. Lobo, R. Bhatia and S. Naqvi, "A Policy Description Language," *AAAI Innovative Applications of Artificial Intelligence*, AAAI Press, pp. 291–298, 1999.
32. Y. Long, H. Lam and S. Su, "Adaptive Grid Service Flow Management: Framework and Model," *IEEE International Conference on Web Services*, IEEE Computer Society Press, pp. 558–565, 2004.
33. H. Ludwig, A. Keller, A. Dan, R. King and R. Franck, *Web Service Level Agreement (WSLA) Language Specification, Version 1.0, Revision wsla-2003/01/28*, <http://www.research.ibm.com/wsla/documents.html>, 2003.
34. H. Ludwig, A. Dan and B. Kearney, "Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements," *ACM International Conference on Service Oriented Computing*, M. Aiello, M. Aoyama, F. Curbera and M. Papazoglou (Eds.), ACM Press, pp. 65–74, 2004.
35. L. Lymberopoulos, E. Lupu and M. Sloman "An Adaptive Policy-Based Framework for Network Services Management," *Journal of Network and Systems Management*, Plenum Publishing Corporation, vol. 11, no. 3, pp. 277–303, 2003
36. V. Machiraju, J. Rolia and A. van Moorsel, "Quality of Business Driven Service Composition and Utility Computing," *HP Labs Technical Report HPL-2002-66*, Hewlett Packard Laboratories, 2002.
37. V. Machiraju, A. Sahai and A. van Moorsel, "Web Services Management Network: An Overlay Network for Federated Service Management," *IFIP International Symposium on Integrated Network Management*, G. Goldszmidt and J. Schönwälder (Eds.), Kluwer, pp. 351–364, 2003.
38. C. Molina-Jimenez, S. Shrivastava, J. Crowcroft and P. Gevros, "On the Monitoring of Contractual Service Level Agreements," *IEEE International Workshop on Electronic Contracting*, IEEE Computer Society, pp. 1–8, 2004.
39. C. Molina-Jimenez, S. Shrivastava, E. Solaiman and J. Warne, "Run-time Monitoring and Enforcement of Electronic Contracts," *Electronic Commerce Research and Applications*, Elsevier, vol. 3, no. 2, pp. 108–125, 2004.
40. N. Muller, *Focus on OpenView: A Guide to Hewlett-Packard's Network and Systems Management Platform*, CBM Books, 1996.

41. G. Nudd and S. Jarvis, "Performance-based Middleware for Grid Computing," *Concurrency and Computation: Practice and Experience*, John Wiley and Sons, vol. 17, pp. 215–234, 2005.
42. T. Nowicki, M. Squillante and C. Wu, "Fundamentals of Dynamic Decentralized Optimization in Autonomic Computing Systems," *Self-Star Properties in Complex Information Systems*, O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel and M. van Steen (Eds.), Springer, LNCS vol. 3460, to appear, 2005.
43. *OASIS Web Services Distributed Management Technical Committee*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm.
44. *OpenView Automation Manager*, http://managementsoftware.hp.com/solutions/server/demo_0001_transcript.html.
45. *OpenView Service Desk SLA*, <http://www.managementsoftware.hp.com/products/sdesk>
46. L. Phifer, "SLAs Meet Managed VPNs," *isp-planet.com*, http://www.isp-planet.com/business/slas_for_vpns1.html, 2000.
47. P. Pongpaibool and H. Kim, "Providing End-to-End Service Level Agreements Across Multiple ISP Networks," *Computer Networks*, Elsevier, vol. 46, no. 1, pp.3–18, 2004.
48. *ProdexNet*, independent software vendor, <http://www.prodexnet.com>.
49. J. Pruyne and V. Machiraju, "Quartermaster: Grid Services for Data Center Resource Reservation," *HP Labs Technical Report, HPL-2003-228*, HP Laboratories, 2003.
50. Y. Ren, D. Bakken, T. Courtney, M. Cukier, D. Karr, P. Rubel, C. Sabnis, W. Sanders, R. Schantz and M. Seri. "AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects," *IEEE Transactions on Computers*, IEEE Computer Press, vol. 52, no. 1, pp. 31-50, 2003.
51. A. Robinson and D. Lounsbury, "Measuring and Managing End-To-End Quality of Service Provided by Linked Chains of Application and Communication Services," *Workshop on Evaluating and Architecting System Dependability*, 2002.
52. B. Rosenthal, "A Surprising New Study: SLAs Now Have Teeth," *OutsourcingSLA.com*, <http://www.outsourcing-sla.com/surprising.html>.
53. A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel and F. Casati, "Automated SLA Monitoring for Web Services," *IFIP/IEEE Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications*, Springer, pp. 28–41, 2002.
54. C. Santos, X. Zhu and H. Crowder, "A Mathematical Optimization Approach for Resource Allocation in Large Scale Data Centers," *HP Labs Technical Report, HPL-2002-64R1*, HP Laboratories, 2002.
55. J. Skene, D. Lamanna and W. Emmerich, "Precise Service Level Agreements," *International Conference on Software Engineering*, IEEE Computer Society, pp. 179–188, 2004.
56. *Sprint Back Bone SLAs and Measured Metrics*, <http://www.sprint.com/business/support/serviceLevelAgreements.jsp>.
57. V. Tasic, B. Pagurek, K. Patel, B. Esfandiari and W. Ma, "Management Applications of the Web Service Offerings Language (WSOL)," *Information Systems*, Elsevier, to appear, 2005.

58. A. van Moorsel, "The 'QoS Query Service' for Improved Quality-of-Service Decision Making in CORBA," *IEEE Symposium on Reliable Distributed Systems*, IEEE Computer Society, pp. 274–285, 1999.
59. A. van Moorsel, "Grid, Management and Self-Management," *The Computer Journal*, Oxford University Press, to appear, 2005.
60. R. West, "Open-Source Network Monitoring Software," *PC Network Advisor*, www.itp-journal.com, no. 124, pp. 3–6, 2000.
61. A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry and S. Waldbusser, *IETF Request for Comments RFC 3198*, 2001.
62. L. Zhang and D. Ardagna, "SLA-Based Profit Optimization in Web Systems," *ACM International Conference on World-Wide Web*, ACM Press, alternate track paper, pp. 462–463, 2004.