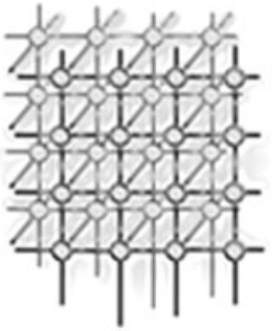
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2007; **19**:1975–1990
Published online 14 November 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1140

Towards increased expressiveness in service level agreements



Viktor Yarmolenko and Rizos Sakellariou*,†

School of Computer Science, The University of Manchester, Kilburn Building, Oxford Road, Manchester M13 9PL, U.K.

SUMMARY

The aim of this paper is to argue for the benefits of increased expressiveness in service level agreements (SLAs). Such benefits may be obtained from the use of analytical expressions to specify a SLA's agreement terms as functions and not as variable or constant values or ranges. The main idea behind this thinking is that functions may contain variables defined in the SLA or be drawn from the known set of reference variables, such as wall-clock time, job start time, current bandwidth of the resource, etc. Experiences and conclusions drawn are in the context of SLA-based job management systems. We demonstrate that the use of analytical expressions in SLAs can potentially reduce the overheads associated with job renegotiation and/or reduce the number of failed agreements. Copyright © 2006 John Wiley & Sons, Ltd.

Received 7 June 2006; Accepted 3 August 2006

KEY WORDS: service level agreement; WS-agreement; job scheduling

1. INTRODUCTION

Developments in the area of service-oriented architectures are expected to attract a range of commercial applications. An important consequence of these developments is the increasing use of the notion of a *service level agreement* (SLA), which is essentially a contract between (usually) two parties, namely the consumer and the provider of a service. This contract outlines the level of quality of service expected and states guarantees for such desired qualities. In the context of Web and Grid services, the current understanding of the community is that such a SLA is essentially an electronic

*Correspondence to: Rizos Sakellariou, School of Computer Science, The University of Manchester, Kilburn Building, Oxford Road, Manchester M13 9PL, U.K.
†E-mail: rizos.sakellariou@manchester.ac.uk

Contract/grant sponsor: The Engineering and Physical Sciences Research Council (EPSRC); contract/grant number: GR/S67654/01

Copyright © 2006 John Wiley & Sons, Ltd.

contract negotiated by and between two processes (machines) and, as such, must be machine readable and understandable. A specification standard for SLAs that has attracted significant interest recently, which has been developed by the GRAAP Working Group of the Global Grid Forum, is known as the *WS-Agreement* (WS-A) [1]. The WS-A specification covers an extensive set of scenarios in general, although some weaknesses have been noticed in the area of SLA renegotiation [2,3].

Among other application domains, SLAs are now being considered in the context of job submission to parallel supercomputer resources [4–6]. Job submission to supercomputer resources has traditionally been queue-based, providing only one level of service, namely 'run this when it gets to the head of the queue'. However, emerging patterns of usage for such resources, arising from Grid computing (and other areas), render queue-based job submission systems ineffective, since they provide only an extreme level of service. The use of SLAs in job submission allows users to specify their own constraints for the earliest possible time to start the execution of a job, the latest acceptable finish time and so on; the owner of the resource can agree, depending on his/her own constraints, in which case a SLA is formed including the terms agreed.

It is anticipated that an efficient system with a level of fault tolerance may include the possibility of renegotiation [7–9] of the whole or part of a SLA. Renegotiation includes reconsideration, by all participants of an agreement, of the quality and the level of service such as time bookings, bandwidth, processor or memory requirements and price. In an earlier study, it has been found that, at high resource load, a significant proportion of agreed SLAs may have to be renegotiated in order to avoid the failure of a SLA [10]. This requires the user's participation at some stage after the time the initial agreement is made and before the work specified by the SLA is completed. Such a requirement might contrast with the objectives commonly set for autonomous systems [11,12], where the aim is to reduce the amount of interaction with the end-user. It is reasonable to say that a degree of autonomicity would be desirable in SLA-based Grid applications. Another related branch—value-added services that are composed of a number of single services—is based on the assumption that a certain level of automation exists.

In this paper, we consider how the impact of renegotiation overheads can be minimized by describing the terms of a SLA in an analytical form using functions. This increases the expressiveness of the SLA and provides more flexibility to the provider of a service in satisfying the terms of the SLA. The scope of this paper does not include negotiation protocols or what XML schema might be used, nor does it consider scenarios other than job submission and management. Its findings directly affect renegotiation for SLA-based resource management, but might also be applicable to other aspects of service-driven applications in service-oriented distributed computing. Thus, the key issue addressed by the paper is how the overhead associated with SLA renegotiation and the user involvement can be reduced, and how all this can fit within an existing framework for SLAs, such as WS-A.

The paper is structured in the following way. Starting with related work in Section 2, it then proceeds with a description of analytically expressed SLA terms (Section 3). This includes a few specific examples (Section 4), and a computer-simulation experiment in Section 5 in which a simple model was constructed in order to evaluate some of the examples presented in Section 4. Concluding remarks are given in Section 6.

2. MOTIVATION AND RELATED WORK

Earlier work [10] indicated that, in SLA-based job-submission systems, at high resource load, renegotiation is inevitable as a means to maximize resource utilization while minimizing the number of

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

failed/cancelled SLAs. In [6], the authors evaluated several SLA-based job management scenarios, all of which use re-planning on local resources. Re-planning involves changes in queue order, nodes used, booked time, etc. and often relies on renegotiation. Renegotiation allows the SLA constraints to be changed while honouring the agreement. This provides the necessary flexibility and achieves a higher resource utilization, although the overall performance suffers as a result. In fact, the performance of the job-submission system might be measured using a function inversely proportional to the number of renegotiations required to satisfy SLAs.

Considering the dilemma between efficient scheduling on the one hand and renegotiation overhead on the other, this paper suggests the use of analytical expressions for SLA terms as a standard. Such expressions already encompass the existing usage of SLAs as a set of constant constraints and provide much more expressive constraints. The latter can potentially reduce the need for renegotiation of a service and provide the extra flexibility needed for a wide spectrum of resource management applications.

In [13], a SLA language specification was defined (WSLA). The WSLA language specification in [13] has a business feel overall. A SLA is treated as no less than a contract with specified obligations on the service provider with respect to the guarantees it provided in the agreement. Also, it specifies the measures to be taken in the case of deviations from or failures to meet such guarantees. In other words, the WSLA defines performance characteristics and an agreed way to evaluate them. As such, WSLA is intended to be used in conjunction with other description languages, such as WSDL [14] for service description or JSDL [15] for description of job submission. Apart from WSLA, the work in [16] concentrated on a language for defining SLA terms using XML. Its aim was to define the semantics of such a SLA language precisely by modelling the syntax of the language in UML. The authors defined a vocabulary for a wide range of Internet services, which was derived from industrial requirements.

The draft of the WS-A specification [1], mentioned earlier, is the product of a collaboration between many scientific and industrial partners. The draft uses XML to describe the WS-A specification, which specifies an agreement between only two parties, a provider and a consumer. Each of the two parties can be either an initiator of or a responder to the agreement. Part of the specification also defines a protocol for the creation of such an agreement using templates. The agreement structure is composed of several distinct parts, namely Name, Context and Terms of Agreement. The latter is also divided in service description terms and guarantee terms. Service descriptions terms mainly describe the functionality to be delivered under the agreement. The guarantee terms define the assurance on service quality for each item mentioned in the service description terms section of WS-A. In the context of job submission, considered in this paper, such assurances may be defined as a parameter (constant) or bounds (min/max) on the availability of part or the whole of the resource. In WS-A, such assurances are referred to as service level objectives (SLOs); in a domain specific to computation services provision, they are usually expressed as values (e.g. SLO : CPUcount = 8). Each SLO may refer to one or more business values, called a business value list (BVL). This list expresses different value aspects of a specific SLO. The other two types of guarantee term are Qualifying Conditions and Importance, which have a similar function to SLO and BVL, respectively.

In this paper we only focus on SLO and BVL, providing examples of how they could be expressed as functions and evaluate the impact of such expressions on the number of times the need to renegotiate a SLA arises. The main idea is to use functions as negotiable values; these values may depend on other negotiable values. Other extensions to the WS-A [2,3] made it possible to label negotiable terms as negotiable, introducing negotiable templates that provided negotiable ranges. However, these extensions still require a number of renegotiations to be performed.

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

3. EXPRESSING SLA TERMS ANALYTICALLY

In most cases of SLA-based resource management applications [17,18], the set of guarantee terms is rigidly defined, even though there is no specific requirement for these constraints to be expressed as constants. Even the WS-A specification, used as a basis for a SLA in [17,18] does not limit SLA terms to constants. Nonetheless, the wide perception for a SLA is that its guarantee and value terms should contain constant values and thus define an agreement in a limited way.

Within such an approach, it is known in advance, for example, what the exact financial gain will be when a SLA is audited. The obvious benefit of such an arrangement is a relatively compact SLA that is simple to understand by a human and has low overheads. However, we find that this arrangement carries too little information for other parties of this SLA to take the initiative for some deviation without the need for renegotiation. The expressive capacity of an agreement can be improved by adding more terms, but there is a limit on how much overhead is practical, whereas most of the terms can easily be described analytically.

We believe, therefore, that in most cases a slightly increased overhead in the SLA can be justified if the number of renegotiations or failed SLAs can be reduced. This reduction can be achieved by providing an infinitely large set of term configurations in the SLA. The extra SLA feeds can provide complex relationships between SLA terms that could potentially be used by autonomous applications in providing, qualitatively, new levels of service. In other words, we propose to express a SLA in such a way that it describes a service with a higher degree of expressiveness. These suggestions are stated below.

(1) To introduce a list of *universal* variables. These can be either constants or static functions, i.e. invariant to the environment. They are predefined elsewhere and simply referred to in the SLA. Among such variables/functions are: current wall-clock time, current network bandwidth, etc. (the relevance of these will be explained shortly).
(2) To introduce a list of predefined common functions. For example, the average value of a list, a Gaussian or other function that defines min/max bounds, etc.
(3) To describe SLA terms not as constants but as functions of *universal* variables, or other SLA terms. These functions can be defined within a SLA or can be one of a set of predefined common functions.

Thus, the terms of a SLA are no longer described as a single point or flat limits (formed by range-values independent of each other) in SLA space. Instead, each SLA term can be expressed as a non-trivial multidimensional function or a system of functions. Such a function can contain an infinite number of acceptable values, which are still within otherwise rigid constraints. Such an agreement has an infinitely large number of outcomes that belong to the continuum defined by the system of functions. When such a system of functions, which describes the entire SLA, is chosen correctly it potentially can provide a far richer term set for job management applications and, hence, reduce the need for renegotiation.

3.1. Universal variables and functions

Earlier, we expressed the need for *universal* variables, the actual values of which may not be known at the time when a SLA is formed. We believe that a SLA specification should be able to implement

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

the notions of such *universal* variables and common functions. Below we give an example of what the SLA vocabulary of these variables and functions may contain.

- *Current time*: returns the global wall-clock time. One could envisage a SLA term which is expressed as a function of current time; for example, a service provider may be encouraged financially to notify the client about any changes as soon as possible, in which case the reward for the notification can be expressed as a function inversely proportional to current time.
- *Current resource load*: returns a value $\{0 \ldots 1\}$ that indicates the current resource load at the time of usage. In the simple case, this value is formed as the ratio of non-idle CPU nodes of the resource to its total number. A service provider may use this value in its price function, where the value of the resource usage is proportional to its load.
- *Current actual bandwidth*: returns the actual bandwidth capability at the time of usage that was allocated to satisfy a specific SLA. This may depend on the time of the day, and other factors. This variable is another example of how a soft boundary can be used in an SLA. The client and the provider may agree on the price for the bandwidth provided; the price varies with the bandwidth capacity in a certain way, for example limited by hard low and high constraints.
- *Actual data traffic*: returns the amount of data that were transferred as a result of the client's job. This term will be discussed in Section 4.
- *Actual disk usage*: returns the amount of disk space used in the client's job. Similar to the previous case, the value returned by this variable, which is unknown at the time of agreement, may be included in one or more SLA terms.
- *Actual maximum memory used*: returns the highest level of memory occupied by the client's job. There are number of possible uses of this term in a SLA. For example, the provider may want to include a clause where it has a right to cancel the job if the memory usage exceeds a certain amount.
- *Actual execution time*: returns the time it took for the client's job to be executed. It is not surprising that the reserved time for the job may be less or more than the actual processing time. Combined with other SLA terms, such as reserved time, SLA participants can create a complex agreement that states discounts for unused time and/or premium rates for overtime. This agreement can be expressed as a single function.
- *Actual job start time*: returns the actual global wall-clock time at which the client's job began its execution. This term will be used in the example in Section 4.1 where detailed reasons for its use are given.

Below we give a list of common functions. The criteria that may determine which functions should be included in the list must be based on: (i) how elaborate the function is in its description; and (ii) how common this function is expected to be.

- $f_{\text{norm}}(x, low, high)$: a binary function which returns 1 in the region $low < x < high$ and 0 for other values of x. Such a function is required to provide the type of constraint which is commonly used in SLAs today. This is another way of determining whether x is within or outside the specified limits. For example, the job must be executed at a time t_S which is not earlier than some time T_S and not later than some time T_F, i.e. $f_{\text{norm}}(t_S, T_S, T_F)$.
- $f_{\text{inv}}(x, low, high)$: a binary function which returns 0 in the region $low < x < high$ and 1 for other values of x. As in the case of $f_{\text{norm}}(\,)$, this function describes the outside ranges of a one-dimensional continuum (e.g. time) limited by two bounds.

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

- $f_{\mathrm{tr}}(x, low, \alpha, high, \beta)$: a function that can be seen as a generalization of $f_{\mathrm{norm}}(\,)$, in the sense that it may allow for intermediate values between 0 and 1, for some x between *low* and *high*. For those x, the value of the function is linearly dependent on x. The shape of the function can resemble a trapezium.

An example using such functions is given in Section 4.1.

Note that the lists of *universal* variables and common functions presented above are not necessarily limited to those elements that have been included here. Also, it may be worth re-emphasizing that the *universal* variables and common functions are predefined and as such they are available offline; thus, there is no need for their descriptions to appear in the SLA, but only in their references. This constitutes the first part of the proposal on how SLAs should be expressed.

3.2. SLA guarantee terms as functions

In the previous section we described some of the *universal* variables and functions to appear in the SLA vocabulary in order for them to be used when defining an arbitrary SLA term, such as price, duration of the job and so on. SLA terms, expressed as a function of such variables may be used, in turn, as part of another function that describes another SLA term. For example, the SLA term that defines the duration of a job may depend on resource bandwidth, which may also depend on resource load, whereas the price for a service may depend on all of these. In this paper, we concentrate only on a few such SLA terms, a breakdown of which follows in the next section.

Furthermore, we present three specific examples of how and why guarantee terms, such as the number of CPU nodes, memory or time slot required for the successful execution of a job, can and should be able to be expressed as functions and not as constants, renegotiable constants or ranges. Similarly, business terms such as price, penalty for the service or importance of a guarantee term can and should also be able to be expressed as functions of any guarantee term(s) as well as *universal* variables and common functions. This constitutes the second and main part of the proposal on how SLAs should be expressed.

4. EXAMPLES

Let us now consider a few simple examples of how a SLA term can be described as a function of one or more other SLA terms, *universal* variables, etc., and when such representations of SLA terms could be useful. First, let us define the SLA terms that will be used in these examples.

- T_{S}: earliest possible time for the job to start. Such a constraint may be one of a number of conditions that must be satisfied before the job can be executed.
- T_{F}: latest acceptable time for the job to finish execution. This constraint may be due to a client's preferred deadline or be part of the same set of conditions as T_{S}, formed as a result of a complex workflow schedule.
- N_{CPU}: number of CPU nodes required for the execution of a parallel job.
- t_{D}: reserved or projected processing time required for job execution. The idea behind this guarantee term is that the client provides certain information about the job (similar to N_{CPU}) as well as guarantees that the job will not overrun the specified time.

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

- B: bandwidth of the resource. Bandwidth is an important parameter that is often specified in the agreement. Its uses range from satisfying the requirements of a specific real-time or streaming application to being an influencing factor on other guarantee terms. (An example of the latter use will follow shortly.)
- D: traffic generated by the parallel job. In order to schedule resources correctly with a level of guarantee as well as price in a fair way it may be important to state the amount of traffic that will be created by the submitting job. (An example of the latter use will follow shortly.)
- $V_{\mathrm{pn}}^{\max}$: limit on the amount of penalty paid by the service provider in the case of a failed or cancelled agreement. In the simplest case, this term is used as a fixed penalty for any failure, but may contain a tree of penalties that correspond to specific failures such as bandwidth, CPU and disk access or, as will be shown later, it may be used as a variable that is used to construct an integrated price/penalty function. In the examples used in this paper, this value is measured in some mutually agreed reward unit per time unit (e.g. \$, £, pigs per hour, etc.).
- $V_{\mathrm{pr}}^{\max}$: limit on the price paid by the client for the provided service. As in the case of $V_{\mathrm{pn}}^{\max}$, this term can be either a price or a maximum price the client pays in order to constrain an integrated price function that may vary in a non-trivial way. Similarly to the previous SLA term, this value is measured using some mutually agreed reward unit per time unit.
- V_{tot}: integrated reward. In the example which will be described shortly, we show that it is possible to design a single SLA term representing price/penalty value as a function of many other SLA terms.

Note that we omit the other fields normally found in SLA implementations such as WS-A, namely, Context, Name and all of the tags associated with them, and only concentrate on a few guarantee terms for simplicity.

4.1. An example with the reward term as a function

Before introducing analytical expressions, we consider a conventional scenario in which there are no functions but SLA terms expressed as normal values (constants). In this scenario the client agrees to pay a fixed amount (equal to $V_{\mathrm{pr}}^{\max} \times t_{\mathrm{D}}$, for consistency with future examples and Section 4) for the job that we assume will take no longer than t_{D} and will be executed within certain time limits (T_{S} and T_{F}). If the duration of the job exceeds t_{D}, then the provider has the right to cancel the job and demand a fixed amount to be paid by the client. If, however, the provider misses the deadlines set by the client (e.g. the job does not finish before T_{F}), then the client may demand an agreed fixed penalty (equal to $V_{\mathrm{pn}}^{\max} \times t_{\mathrm{D}}$, for the same reason as above) to be paid by the provider.

Let us now introduce analytical expressions into this case. In order to describe the same scenario, we can use the predefined function $f_{\mathrm{norm}}(\,)$ so that the integrated reward (which includes both outcomes described earlier) will be

$$V_{\mathrm{tot}} = (f_{\mathrm{norm}}(t_{\mathrm{S}},\, T_{\mathrm{S}},\, (T_{\mathrm{F}} - t_{\mathrm{D}})) \cdot (V_{\mathrm{pr}}^{\max} + V_{\mathrm{pn}}^{\max}) - V_{\mathrm{pn}}^{\max}) \cdot t_{\mathrm{D}} \tag{1}$$

where t_{S} is the actual start time of the job (the remaining terms are SLA terms described in Section 4). In other words, if the provider runs the job between T_{S} and T_{F}, it gets paid $V_{\mathrm{pr}}^{\max} \times t_{\mathrm{D}}$; if it misses the deadline set by the client, then the client receives $V_{\mathrm{pn}}^{\max} \times t_{\mathrm{D}}$ as compensation.

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

This seems like too much trouble with what is an otherwise simple agreement. However, let us now consider a scenario in which the parties agree to a gradually increasing penalty, depending on how late the provider is. Again, this can be specified using constants in SLA terms, but the size of the agreement would grow as more fine-grained penalty dependence is required. Let us now look how a business term such as the integrated reward can be described in a more laconic way—as a function of other SLA terms, *universal* variables, etc.

We start building the function for the V_{tot} SLA term by specifying the boundaries using $f_{\text{norm}}(t_S, T_S, T_F)$ (see Section 3.1). The graphical representation of this function is shown in Figure 1(a). Now, we introduce the maximum price and penalty factors, $V_{\text{pr}}^{\max}$ and $V_{\text{pn}}^{\max}$, into the equation, so that

$$f'_{\text{norm}} = (f_{\text{norm}}(t_S, T_S, T_F) \cdot (V_{\text{pr}}^{\max} + V_{\text{pn}}^{\max}) - V_{\text{pn}}^{\max}) \tag{2}$$

Figure 1(b) shows how exactly the SLA terms $V_{\text{pr}}^{\max}$ and $V_{\text{pn}}^{\max}$ change the view of the binary step function f_{norm}. The function f'_{norm} differs from f_{norm} in that instead of returning 0 or 1 it returns $V_{\text{pr}}^{\max}$ or $V_{\text{pn}}^{\max}$ depending on the value of t_S. Note that Equation (2) does not contain any dependence on job duration. Unlike Equation (1), the integrated reward, V_{tot}, is going to be expressed as the sum of fixed price units per time unit over a period of time t_D and its analytical view will be as follows:

$$V_{\text{tot}} = \sum_{t=t_S}^{t_S+t_D} f'_{\text{norm}} \tag{3}$$

Figure 1(c) illustrates the result of Equation (3) as a shaded area defined by the function f'_{norm} and the limits t_S and $t_S + t_D$. In this exercise, we assume that t_D is equal to the actual time it takes for the job to be completed; however, in cases where t_D might be different, Equation (3) would provide a more honest reward value for the service compared with a fixed price SLA alternative. If the job overruns the projected time t_D, the agreement can still be honoured.

Now, let us look at the case when the provider starts the job too late for it to be completed before the agreed deadline T_F (Figure 1(d)). The provider will be paid in full for the time between the start of the job and T_F, but, after time T_F, it incurs a penalty that will reduce the overall reward specified in Equation (3) by adding a negative quantity to the total sum. In this particular case (Figure 1(d)), the provider would still be in credit as it appears that $V_{\text{pr}}^{\max} > V_{\text{pn}}^{\max}$ and the job spent an equal amount of time within and outside the deadline.

We have shown in a simple example how using a simple expression for a SLA term can cover a higher degree of possible outcomes. Using the area specified by f'_{norm} is one way of increasing the expressiveness of the constraints. This increases the chances that a SLA will not need to be renegotiated (or cancelled), still retaining the intended preferences. On the other hand, choosing f'_{norm} as an expression for the integrated reward simply falls back to a conventional rigid or less expressive SLA ($V_{\text{tot}} = f'_{\text{norm}}(t_S, T_S, (T_F - t_D))$, to be precise); this means that rigid constraints can be expressed by functions just as well. Such expressive constraints more accurately characterize business practices and contracts today. Looking forward, the expressive SLA can be viewed as a tool for autonomous renegotiation where the SLA itself can serve as a representative of both parties, providing a new set of terms when queried, and not merely a record of an agreement.

To reiterate the potential of an expressive SLA, let us build a slightly different, more complex, integrated reward function. We start by changing the way the time constraints (T_S and T_F) influence the integrated reward V_{tot} from a step form in the previous example to a more elaborate function using

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

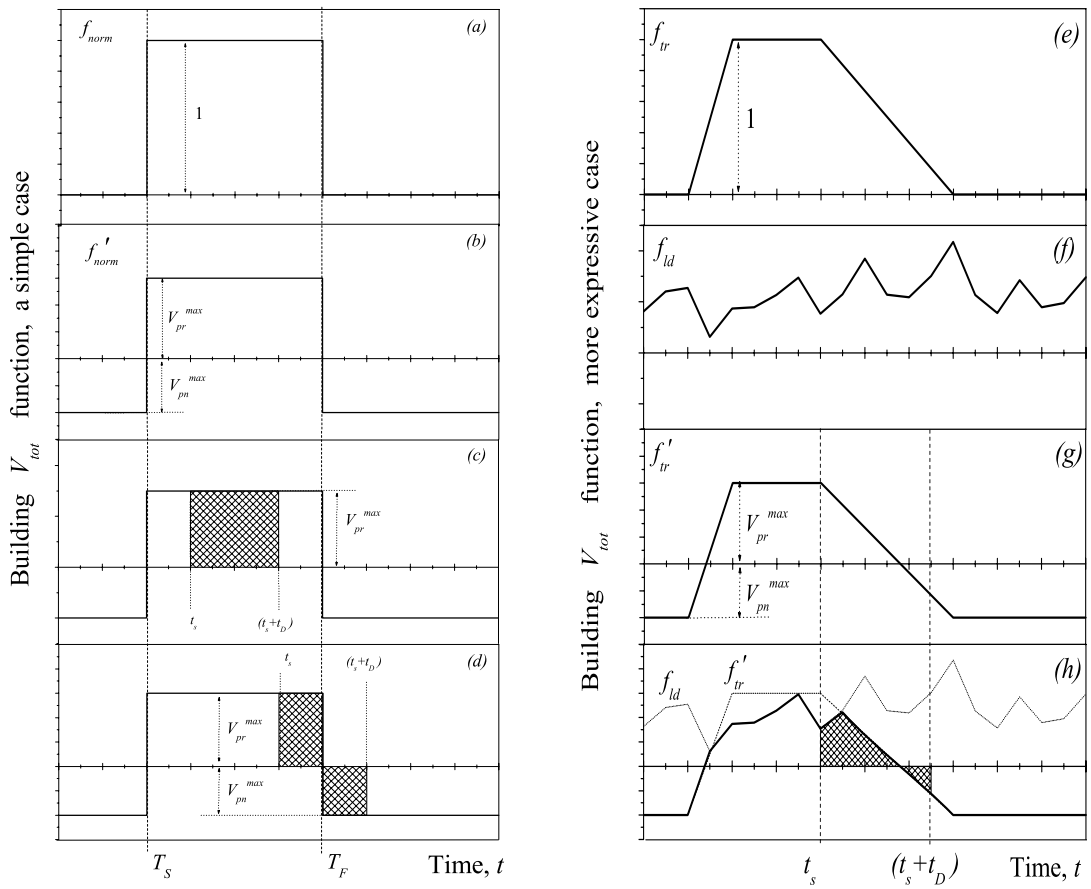


Figure 1. An example of integrated reward, V_{tot}, expressed as a function (Equations (3) and (5) are built using functions from graphs (a), (b), (c), (d) and (e), (f), (g), (h), respectively). (a) f_{norm} a binary step function (Section 3.1); (b) f'_{norm} function (Equation (2)); (c) and (d) a resulting area under f'_{norm} limited by parameters t_{S}, t_{D} is an integrated reward value calculated from Equation (3); (e) f_{tr} function (Section 3.1); (f) the resource load function (Section 3.1); (g) a version of (a) that defines values for price/penalty (Equation (4)); (h) same as (g) but takes into account the price variation depending on (f).

the function f_{tr} introduced in Section 3.1. A graphical view of this function is shown in Figure 1(e). We then introduce the maximum price and penalty factors, $V_{\text{pr}}^{\max}$ and $V_{\text{pn}}^{\max}$, in exactly the same way as in the previous case. The new function f'_{tr} now becomes

$$f'_{\text{tr}} = (f_{\text{tr}}(t_{\text{S}}, T_{\text{S}}, \alpha, T_{\text{F}}, \beta) \cdot (V_{\text{pr}}^{\max} + V_{\text{pn}}^{\max}) - V_{\text{pn}}^{\max}) \tag{4}$$

where α, β are the parameters of f_{tr} that determine the slope of the two sides of the trapezium (Figure 1(g)). As in the previous case, the area of the function below zero (as low as $V_{\text{pn}}^{\max}$) will contribute to the penalty whereas the area above zero (as high as $V_{\text{pr}}^{\max}$) contributes to the price.

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

At this stage, we would like to point out the time limits of V_{tot}, namely the start time of the job t_{S} and its duration t_{D} (Figure 1). We consider only one case in which the job starts at t_{S} (Figures 1(g) and (h)). We still assume that the time reserved for the job, t_{D}, is equal to the actual time it took for the job to be completed, although one could introduce a relationship between these terms, which might influence the final reward.

Thus far, we have introduced SLA terms that are usually defined by the client, with the exception of t_{S}. We now introduce a constraint on V_{tot}, which is imposed by the service provider. For example, consider a function that represents the resource load, $f_{\text{ld}}(t)$ (Section 3.1 and Figure 1(f)). The provider may want to vary the amount of the reward generated from the service depending on the resource load (demand for this service); that is, if the resource is overloaded, the provider may charge more. Naturally, $f_{\text{ld}}(t)$ is unknown when the agreement is created.

If we represent time in this example as a continuous variable (a more realistic view of time as opposed to the previous example), we build V_{tot} so that only the intersection of the area specified by each function, $f_{\text{ld}}(t)$ and f'_{tr}, contributes to the outcome (Figure 1(h)):

$$V_{\text{tot}} = \begin{cases} \displaystyle\int_{t_{\text{S}}}^{t_{\text{S}}+t_{\text{D}}} f'_{\text{tr}}(t_{\text{S}}, T_{\text{S}}, \alpha, T_{\text{F}}, \beta, V_{\text{pn}}^{\max}, V_{\text{pr}}^{\max})\, \mathrm{d}t \\ \displaystyle\int_{t_{\text{S}}}^{t_{\text{S}}+t_{\text{D}}} c \cdot f_{\text{ld}}(t)\, \mathrm{d}t \end{cases} \tag{5}$$

A weighting coefficient, c, is used to adequately represent the resource load with respect to the $f'_{\text{tr}}(\,)$ function. The shaded area represents the total reward, V_{tot}, that a client has agreed to pay. In the case where t_{S} occurs at a point in time later than that shown in Figure 1(h), the value of V_{tot} decreases to the point where the shaded area below zero is larger than that above zero; the service provider then pays some penalty, but not higher than the limit imposed by $V_{\text{pn}}^{\max}$.

Thus, we have defined a SLA term in the agreement which is based on functions and *universal* variables whose values were unavailable at the time when the SLA was formed. Also, in this example, we were able to agree on an infinitely large number of outcomes for V_{tot}, using the notion of continuous time. Now, the service provider can re-plan its activities without the need for renegotiation of a new schedule with the client.

Similarly, other business and guarantee terms can be described as functions or a system of functions. For example, a client's application can start with different memory or CPU requirements, which may affect the execution time, with the relationship defined in term t_{D}. Alternatively, a service provider may want to encourage clients to use wider time bounds, so that the business term of a service may depend on the time window, $(T_{\text{F}} - T_{\text{S}})$, specified by the client.

4.2. An example with variable number of CPUs

Another simple example of usage of an expressive SLA would be useful for applications that may need to vary the number of CPU nodes required at the start of a computation (e.g. MPI-based parallel applications). The expressive SLA would look similar to the standard SLA, with the exception of three of its guarantee terms, namely:

- maximum number of CPU nodes to be used, $N_{\text{CPU}}^{\max}$;
- number of CPU nodes required, $N_{\text{CPU}} = \{2, 3, 4, \ldots, N_{\text{CPU}}^{\max}\}$;
- reserved time for job execution, $t_{\text{D}} = t_{\text{D}}^{1}/N_{\text{CPU}}$;

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

where t_D^1 is the projected time for the job to be completed on a single CPU node and N_{CPU}^{max} is limited by the capacity of the resource in this example. The function for t_D can be more complex than that presented here. Compared with the last example in the previous section, the function presented here can only produce a limited number of options. However, even in a case such as this, a relatively simple function could describe something in a more laconic fashion than a list of paired values (N_{CPU} and t_D), which, in the simplest case, can reach $(N_{CPU}^{max} - 1)$ items in size (the option where the value of N_{CPU} is equal to 1 is not considered). The number of possible renegotiations even for such a limited set is unacceptably high. If the resource's scheduling algorithm needs to renegotiate every single $\{N_{CPU}, t_D\}$-paired option for each job submitted in order to increase its utilization, then, potentially, $(N_{CPU}^{max} - 1)^{N_{job}}$ renegotiations would be required (assuming that responses are cached by the resource).

Owing to its relevance and simplicity, we ran an experiment (see Section 5), which quantitatively measured benefits of an expressive SLA in which N_{CPU} and t_D correlate. The results show a significant increase in resource utilization when the resource load is greater than about 30%. Summarizing this example, it is worth adding that a user may want to add variable importance as to what number of CPU nodes to use, adding extra correlation to the business SLA terms. This would combine the features of this example with those presented in the example described in Section 4.1.

4.3. An example with variable bandwidth

The next example considers the speed of communication between the working nodes. For example, for message-passing-intensive applications, a 32-processor machine is likely to perform better than a cluster that aggregates 32 individual nodes of comparable power spread across the world. As in the previous examples, let us define the relevant terms of agreement:

- bandwidth between the nodes, B;
- traffic generated by the parallel job, D;
- reserved time for job execution, $t_D = t_D^\infty + D/B$;

where t_D^∞ is the time it takes for the job to be completed on an infinite bandwidth parallel resource.

Again, the relation for t_D can be more complex than that. In general, the more detailed the description, the better the service that can be achieved.

4.4. Examples summary

In the previous sections we gave three different examples of how analytical expressions can be used effectively in SLAs. Combining the guarantee terms from these examples into a single SLA increases the number of options available to the resource, making value added autonomous services more feasible. In fact, the aggregated view of the new SLA terms for the three examples will be as follows:

- bandwidth of the resource, B (universal value; may be a function of time);
- traffic generated by the parallel job, D (universal value; it is unknown at SLA creation);
- duration of a parallel job, $t_D = t_D^1/N_{CPU} + D/B$, where t_D^1 is the projected time for the job to be completed on a single CPU node ($t_D^\infty = t_D^1/N_{CPU}$);
- limit on the amount of penalty carried by the service provider, $V_{pn}^{max} = \text{constant}$;
- limit on the amount of value paid by the client, $V_{pr}^{max} = \text{constant}$;
- integrated reward, V_{tot}, see Equation (5).

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

The SLA described above achieves a certain level of mathematical complexity, which might be more error-prone if human beings were to generate and negotiate the SLAs. However, the use of functions serves exactly the opposite purpose, which is to make the negotiation, analysis and generation of SLAs more easily amenable to machine processing. In summary, we would like to add that we do not advocate the maximum complexity in the description of the relationships between the SLA terms. We merely suggest that the way in which a SLA is described should not be restrictive in case the need for higher complexity eventually arises.

5. EXPERIMENTS AND RESULTS

This experiment refers to the example described in Section 4.2 and is built on earlier work [19]. The aim is to evaluate the negotiation efficiency of a system that uses expressive SLAs, as opposed to a system that uses conventional SLAs, focussing on the job load. We choose two metrics: average job rejection rate and average number of renegotiations per job request. The model description and relevant details follow.

Two parties are involved in the SLA, the user and the resource. The resource schedules jobs using a single iteration earliest T_F-deadline first algorithm. Note that scheduling performance is beyond the scope of this paper; a description of several heuristics for scheduling is given in [20]. The availability of the resource is 64 working nodes over 600 virtual hours. The user always generates a set of job requests in which at least one configuration exists, by which jobs can be scheduled on the resource with 100% utilization, so that 100% of SLAs are satisfied with zero idle resources during the availability period. The job requests vary in their parameters and also differ from set to set. The number of requests in each set varies from 600 to 900. The job requests in each set are generated randomly following a Gaussian distribution in which the product of terms N_{CPU} and t_D (or term t_D^1, see Section 4.2) peaks at around 37 CPU-hours for all job sets generated. The time limits T_S and T_F varied from job to job and differed in each set. These were generated randomly following a Gaussian distribution in which the difference $(T_F - T_S)$ peaked at around 40 virtual hours. The number of different job sets used in this experiment was 10.

When the request to submit a job was successful, a SLA was formed between two parties containing the guarantee terms T_S, T_F, N_{CPU} and t_D. Other terms (including business terms) discussed in earlier examples are omitted in this experiment for simplicity. In each SLA, the term t_D is expressed as a function of N_{CPU} (see Section 4.2). Only the cases with integer values of N_{CPU} and t_D were considered. For example, for $t_D^1 = 24$, the relation $t_D = t_D^1/N_{CPU}$ can produce only seven options for the pair $\{t_D, N_{CPU}\}$ (the option where N_{CPU} is equal to 1 is not considered). Depending on the value of t_D^1, the number of options vary from SLA to SLA with the average number of options across all job sets being below seven.

The first scenario represents job submission using a conventional SLA. In this scenario, all SLA terms are represented by constants. A constant value for N_{CPU} is chosen randomly from the set of available options and is fixed throughout the entire simulation. In the simulation, the user attempts to submit a job (including the constraints that may be used to form the SLA terms) one by one to the resource. If the resource is able to fit the job, then a SLA is formed. If the resource refuses to accept it, the job is discarded.

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

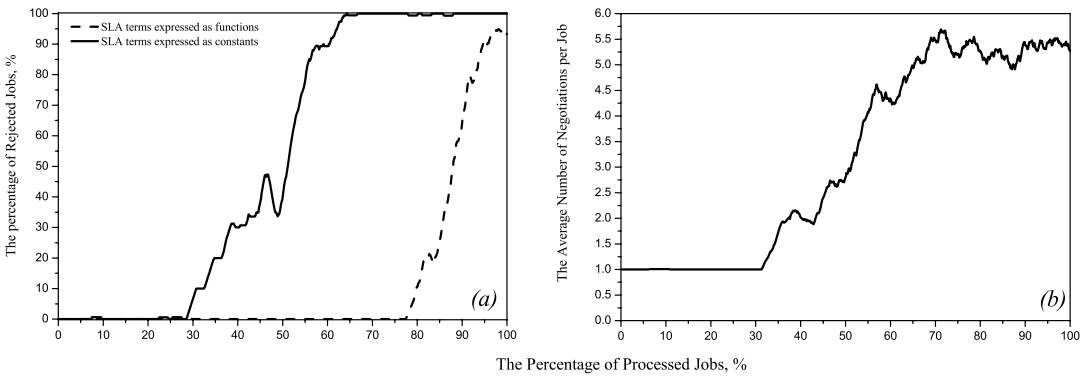


Figure 2. (a) The rate of cancelled job requests depending on the percentage of jobs submitted. Job submission using expressive SLAs (dashed curve) produced less rejected jobs at high resource load compared with the conventional SLA scenario (solid curve). (b) The rate of negotiation attempts depending on the percentage of jobs submitted.

The second scenario differs from the former in that it uses the function mentioned above to describe t_D. The initial configuration in this scenario also starts with a random value of N_{CPU}. However, if the resource cannot accept the job with these parameters, it simply tries another configuration defined by the expressive SLA. This scenario represents job submission using a SLA with terms defined as functions; in this specific example, there is only one term (t_D) that is described in such a way, and the number of possible options generated from this is limited. In real life, t_D may be more fine-grained than the t_D of the simulation, resulting in more options for the relevant SLA terms (see Section 4.2).

In Figure 2(a), we present two curves that were obtained using the scenarios mentioned above. The values were averaged over 10 different job sets. As we can see from the figure, the scheduling algorithm begins to struggle as the resource load increases and begins to reject more of the requests as a result. In the case when the SLA describes terms as fixed values, the resource rejects a higher number of jobs, on average, than the number rejected in the case of an expressive SLA. In fact, when conventional SLAs are used in the negotiation, almost all job requests are rejected at resource loads higher than about 65%. This can be explained by the fact that the SLA contains t_D and N_{CPU} configurations that are unfavourable to the resource, creating gaps that cannot be filled in by other jobs. Using expressive SLAs, on the other hand, boosts the performance of the same resource to a rejection rate of almost zero. The performance of the latter only begins to decrease at resource loads over 80%. This is because the resource now has the capability derived from the expressive SLA, to choose a configuration that will fit in the schedule. An analogy to what happens in this example can be drawn by thinking of a rectangle packing problem, where, in the first case, the dimensions of each rectangle to be packed cannot change, whereas, in the second case, the dimensions can change as long as the total area of each rectangle remains the same. It is reasonable to expect that in the second case it would be possible to pack more rectangles within the same space.

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

We now consider a third scenario; this extends the first scenario by allowing for renegotiation of rejected requests. The job sets are the same as in the previous two scenarios. With respect to the second scenario, the difference is that the resource must contact the user each time the next pair of terms is going to be used.

Figure 2(b) shows the dependence of the average number of renegotiations per job on the percentage of submitted jobs. The number of renegotiations increases as more jobs are submitted to the busy resource. This value remains at 1 at any resource load, when the expressive SLA is used, because the resource has all of the information in the first instance and does not require additional renegotiations. The average number of renegotiations in the figure saturates to the average number of configurations per SLA in a job set. As the resource struggles more to fit a job, it runs through all possible configurations before giving up. It is obvious that the number of renegotiations, limited in this simulation, may potentially reach large numbers, rendering the entire renegotiation process difficult to manage. In real life, the number of options can progress from a limited set to infinity for each guarantee term. Also, more guarantee terms can be represented as analytical functions of other guarantee terms. This could potentially increase the overall efficiency although it may come at the cost of higher overhead. Currently, this issue is addressed by making a compromise; in this situation, only a limited number of renegotiations is allowed before the expense becomes too great.

6. CONCLUSIONS AND FUTURE WORK

This paper has made a case for more expressive SLAs. The case for the expressiveness of SLAs included the introduction of *universal* variables and common functions and expressing business and guarantee terms of a SLA as analytical functions of *universal* variables, common functions and/or other guarantee terms. We showed, using a simple experiment, that even a little flexibility in the agreement, introduced by expressing only one guarantee term, t_D, as a function, significantly enhances the overall system performance. The situation is bound to be improved when the number of configuration options in the SLA approaches infinity, something that it would be possible to do with the analytical expressions proposed in this paper.

We believe that such a form of SLA not only creates measurable benefits for the current job management models, but may also encourage new service models to emerge, especially in the area of value added and autonomous services. Summarizing, we outline the following benefits yielded by expressing SLA terms analytically.

(1) Reduction in the network traffic associated with the negotiation of a service.
(2) Reduction in the user–service interaction, owing to the increased autonomy of the process, which consequently improves the quality of the service (at least for job submission).
(3) An expressive SLA enables better use of optimization algorithms for problems related to job management.
(4) An expressive SLA could support a larger variety of services, especially commercial services geared toward automation and value added approaches.
(5) An expressive SLA could potentially fit as an extension to the existing SLA specifications such as a WS-A.

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

The ideas laid out in this paper were spawned by a project investigating different aspects related to the usage of SLAs in parallel job scheduling; further questions have arisen as a consequence of the work described in this paper. First of all, further development and evaluation of various models or relationships between SLA terms are needed to yield practical benefits for the scientific community and industry. For example, further work can look into how different relationships between terms correlate. Another avenue to explore is to examine how ontologies and associated technologies could be used in the design of interoperable SLAs for autonomous SLA generation, interpretation and management. The function-based approach describes relationships between SLA terms in a more expressive and dynamic way than a parameter-based approach. The SLA terms described by a function could be further organized into interrelating higher-level terms using ontologies.

ACKNOWLEDGEMENTS

This research has been partially funded by EPSRC, U.K. (grant reference GR/S67654/01), whose support we are pleased to acknowledge. We would also like to thank the anonymous referees for their helpful comments on earlier versions of this paper.

REFERENCES

1. Andrieux A, Czajkowski K, Dan A, Keahey K, Ludwig H, Pruyne J, Rofrano J, Tuecke S, Xu M. Web Services Agreement (WS-Agreement) Specification, Version 2005/09. *GWD-R (Proposed Recommendation) to Global Grid Forum*, Global Grid Forum, 20 September 2005. Available at: http://www.ggf.org.
2. Andrieux A, Dan A, Keahy K, Ludwig H, Rofrano J. Negotiability constraints in WS-Agreement, Version 0.1. *Document to GRAAP-WG Meeting*, Argonne, IL, 26–27 January 2004. Open Grid Forum: Lemont, IL, 2004.
3. Dan A, Keahy K, Ludwig H, Rofrano J. Guarantee Terms in WS-Agreement, Version 0.1. *Document to GRAAP-WG Meeting*, Argonne, IL, 26–27 January 2004. Open Grid Forum: Lemont, IL, 2004.
4. MacLaren J, Sakellariou R, Krishnakumar KT, Garibaldi J, Ouelhadj D. Towards service level agreement based scheduling on the Grid. *Proceedings of the Workshop on Planning and Scheduling for Web and Grid Services (in conjunction with the 14th International Conference on Automated Planning and Scheduling—ICAPS-04)*, Whistler, British Columbia, Canada, 3–7 June 2004; 100–102.
5. SLA based scheduling heuristics, 2006. http://www.gridscheduling.org [20 September 2006].
6. Dumitrescu C, Foster IT. GRUBER: A Grid resource usage SLA broker. *Proceedings of Euro-Par 2005* (*Lecture Notes in Computer Science*, vol. 3648). Springer: Berlin, 2005; 465–474.
7. D'Arienzo M, Pescapè A, Romano SP, Ventre G. The service level agreement manager: Control and management of phone channel bandwidth over premium IP networks. *Proceedings of the 15th International Conference on Computer Communication (ICCC'02)*, Bandra, Mumbai, India, 2002. International Council for Computer Communication: Washington, DC, 2002; 421–432.
8. Czajkowski K, Foster I, Kesselman C, Sander V, Tuecke S. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002 (*Lecture Notes in Computer Science*, vol. 2537). Springer: Berlin, 2002; 153–183.
9. Czajkowski K, Dan A, Rofrano J, Tuecke S, Xu M. Agreement-based Grid service management (OGSI-Agreement). *GRAAP-WG Author Contribution*, Global Grid Forum, 12 June 2003.
10. Yarmolenko V, Sakellariou R, Ouelhadj D, Garibaldi JM. SLA based job scheduling: A case study on policies for negotiation with resources. *Proceedings of e-Science All Hands Meeting (AHM2005)*, Nottingham, U.K., September 2005. EPSRC: Swindon, 2005; 20–22.
11. Sterritt R, Hinchey MG. Why computer-based systems should be autonomic. *Proceedings of the 12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005)*, Greenbelt, MD, 3–8 April 2005. IEEE Computer Society Press: Los Alamitos, CA, 2005; 406–414.
12. Kephart JO, Chess DM. The vision of autonomic computing. *Computer* 2003; **36**(1):41–50.
13. Ludwig H, Keller A, Dan A, King RP, Franck R. Web Service Level Agreement (WSLA) Language Specification, Version 1.0. IBM Corporation, 2003. http://www.research.ibm.com/wsla/WSLASpecV12003 [1 October 2005].

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

14. Christensen E, Curbera F, Meredith G, Weerawarana S. Web Services Description Language (WSDL), Version 1.1, World Wide Web Consortium (W3C), 2006. http://www.w3.org/TR/wsdl [20 January 2006].
15. Anjomshoaa A, Brisard F, Drescher M, Fellows D, Ly A, McGough S, Pulsipher D, Savva A. (eds.). Job Submission Description Language (JSDL) Specification, Version 1.0, GFD-R-P.056. Global Grid Forum, Lemont, IL, November 2005.
16. Skene J, Lamanna D, Emmerich W. Precise service level agreements. *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, Edinburgh, Scotland, May 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004.
17. Austin J, Fletcher M, Jackson T. Distributed aero-engine condition monitoring and diagnosis on the GRID: DAME. *Proceedings of the 17th International Congress and Exhibition on Condition Monitoring and Diagnostic Engineering Management (COMADEM 2004 International)*, Cambridge, U.K., 23–25 August 2004. COMADEM International: Birmingham, 2004.
18. Mobach DGA, Overeinder BJ, Brazier FMT. A resource negotiation infrastructure for self-managing applications. *Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC 2005)*, Seattle, WA, 2005. IEEE Computer Society Press: Los Alamitos, CA, 2005.
19. Sakellariou R, Yarmolenko V. On the flexibility of WS-Agreement for job submission. *Proceedings of the 3rd International Workshop on Middleware for Grid Computing (MGC'05)*, New York, NY, 2005. ACM Press: New York, 2005.
20. Yarmolenko V, Sakellariou R. An evaluation of heuristics for SLA based parallel job scheduling. *Proceedings of the 3rd High Performance Grid Computing Workshop (in Conjunction with International Parallel and Distributed Processing Symposium (IPDPS'2006))*, Rhodes, Greece, 25–29 April 2006. IEEE Computer Society Press: Los Alamitos, CA, 2006.

Copyright © 2006 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe