Context-Driven Autonomic Adaptation of SLA

Caroline Herssens[1], Stéphane Faulkner[2], and Ivan J. Jureta[2]

[1] PReCISE, LSM, Université catholique de Louvain, Belgium
[2] PReCISE, LSM, University of Namur, Belgium
caroline.herssens@uclouvain.be, stephane.faulkner@fundp.ac.be,
ivan.jureta@fundp.ac.be

Abstract. Service Level Agreements (SLAs) are used in Service-Oriented Computing to define the obligations of the parties involved in a transaction. SLAs define the service users' Quality of Service (QoS) requirements that the service provider should satisfy. Requirements defined once may not be satisfiable when the context of the web services changes (e.g., when requirements or resource availability changes). Changes in the context can make SLAs obsolete, making SLA revision necessary. We propose a method to autonomously monitor the services' context, and adapt SLAs to avoid obsolescence thereof.

Keywords: SLA, adaptation, service context.

1 Introduction

Web services are a response to growing needs of responsive and configurable applications on the Internet. A service is a self-describing and self-contained modular application designed to execute a well-delimited task, and that can be described, published, located, and invoked over a network [20]. Web services are supported by technologies such as SOAP, UDDI and WSDL [27] and are accessed via a Uniform Resource Locator.

Given the growing number of available web services on the Internet, different service providers may offer services that provide the same functionality to the users. Such competing services can be distinguished by comparison over nonfunctional characteristics, which take the form of Quality of Service (QoS). QoS is a combination of several qualities or properties of a service, e.g., availability, security, response time or throughput [15]. When a user requests a service to perform some given task, a service is selected that fits the user's QoS requirements. The selected service is the one that meets the most adequately user's preferences over quality attributes that go into QoS. Once the service is selected, it is assigned by the definition of a contract that defines a Service Level Agreement (SLA) between the user and the provider [11,17]. SLAs are used to meet user's requirements, manage user's expectations, regulate resources and control costs [22]. In short, SLAs are used to set the QoS level offered by the service provider to the service user; SLAs result from a negotiation initiated between these parties [10].

However, offered and requested QoS may both vary over time. We say in this paper that such variations occur because of changes in the *context* of services.

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 362–377, 2008.

Given that the term "context" can be widely understood, a definition local to this paper is in order: *context* is any information about the interaction between users and a web service, for which an SLA is specified.

Changes in the context should be reflected in the SLA governing the interaction. Both the offered and the requested QoS levels may vary over the course of the interaction. Moreover, there are dependencies across different context elements that indicate a propagation of variations from one element to multiple context elements. To keep the SLA unchanged in such conditions is to make the SLA obsolescent.

Contributions. We propose an approach that enables an autonomic adaptation of SLA to respond to occuring context modifications. We illustrate our approach with a case study based on European Space Agency (ESA) services used to process information provided by the Envisat satellite. We provide conceptual bases necessary for SLA adaptation. We classify context elements into five distinct categories: user, provider, resource, environment and web service. We also introduce dependencies existing between elements of context enabling to propagate context modifications. We then present our SLA adaptation approach. We propose an architecture relying on an SLA manager to drive the autonomic adaptation based on context elements. Our adaptation uses context modifications and dependencies to enable an autonomic adjustment of existing SLAs to ensure the service conformance to user expectations. Adaptation involves the following steps:

1. Context modifications are reported to the SLA manager that identifies changes and starts the adaptation process.
2. Observed context variations are propagated through context dependencies existing over different elements of context by the SLA manager.
3. Once context variations have been propagated to all context categories, the SLA manager checks the compatibility between user expectations and provider capabilities.
4. Upon base of the result of the compatibility checking, the SLA manager keep the existing SLA, set up a new SLA between the user and the same provider or select another service fitting better to user expectations.

Organization. Section 2 introduces the ESA case study used throughout the paper. Section 3 presents the conceptual elements used to drive the SLA autonomic adaptation and illustrate these concepts with the case study. Section 4 propose our SLA management architecture and assesses the different steps of our adaptation process. The case study illustrates the adaptation process. Section 5 presents the related work; Section 6 draws conclusions and outlines future work.

2 Case Study

The European Space Agency (ESA) program on Earth observation allows researchers to access and use the infrastructure operated and the data collected by

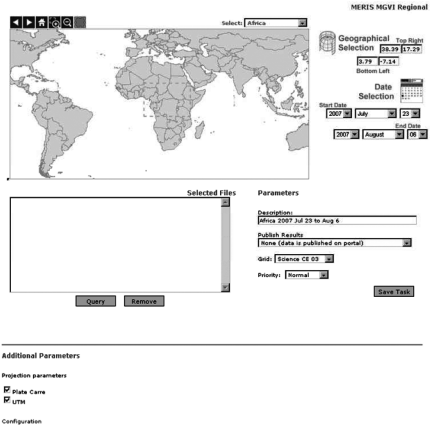


Fig. 1. Graphical interface of the MERIS/MGVI service

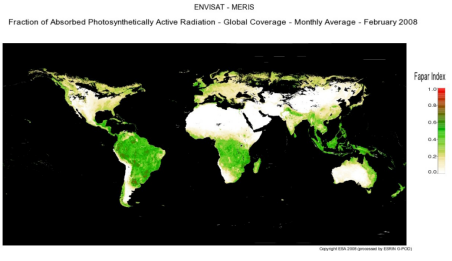


Fig. 2. Output of the MERIS/MGVI service

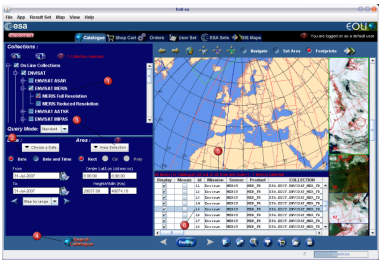


Fig. 3. Graphical interface of the EOLI-SA service

the agency[1]. Our case study focuses on the information provided by the MERIS instrument of the ENVISAT satellite. MERIS is a programmable, medium-resolution imaging spectrometer operating in the solar reflective range. MERIS is used in observing ocean color and biology, vegetation and atmosphere and in particular clouds and precipitation. In relation to MERIS, a large set of web services is made available by the ESA for access to the data the instrument sends and access to the provided computing resources.

We are interested in the remainder about two specific services. The first provides the vegetation indexes for a given period of time and region of the world. A vegetation index measures the amount of vegetation on the Earth's surface. The graphical interface used by the requester of the service is shown in Figure 1. An illustration of the output provided for the whole world map is given in Figure 2. The data on the vegetation index can be obtained for any time range and it is possible to delimit the region of the world that is of interest. This service is

[1] `http://gpod.eo.esa.int`

subject to particular nonfunctional properties: the latency is initially situated between 4 and 6 hours by day of the selected period due to the quantity of data to process. Thus, the service user expects to minimize the execution time. For a service facing such significant latency, service reliability is another critical QoS aspect. Indeed, in case of failures, all execution steps must be started over. So, maximizing the reliability reduce risks to have to start over. The second web service used to illustrate our approach is the EOLI-SA service: this service is used to calculate metadata on the products to process. For example: when you submit a zone to process the MGVI with the 'bounding box' argument, these coordinates need to be transformed into the technical data of the satellite at the time of the acquisition of the zone to process (start/stop time, orbit, lat/long, azimuth angle, etc.). The graphical interface of the EOLI-SA service is given in Figure 3. This service presents different nonfunctional characteristics: while it is used by other services as the MERIS MGVI/Regional, the availability of this EOLI-SA must be maximized in order for these services to execute successfully.

3 Conceptual Foundations

This section introduces the concepts used to drive our autonomic SLA adaptation. We present our notion of context, and subdivide it into categories in Section 3.1. Section 3.2 introduces the concept of dependencies over context elements. All the concepts are illustrated through services introduced in Section 2.

3.1 Context Categories

Unexpected events can modify the current execution context and have an impact on the performance of the services. These modifications can breach the SLA. To adapt existing SLAs to context modifications, context elements need to be accurately defined by services providers and users. We classify context elements into several categories, shown schematically along with potential between-category interactions in Figure 4.

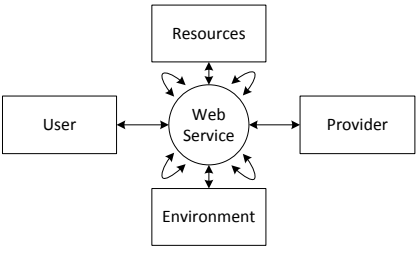


Fig. 4. Context categories and between-category interactions

User context. The *user context* covers the user's QoS requirements. These requirements are expressed with help of preferences over QoS values that the

service must achieve but also with QoS priorities specifying which QoS properties will be maximized over others [21]. The user context also carries information on past executions of services, along with advertised and observed QoS values during these executions [16]. Changes in user context may eventually induce the definition of new SLAs between the user and the provider. The user specifies and updates the user context.

Resources context. Web services executions are influenced by the availability of the resources that concern the network connection between the provider and the user but also the hardware used in executing the service and/or retrieving its results [17]. It is clear that resource availability has a direct impact on delivered QoS, thereby affecting the satisfaction of SLA. Both the service user and the service provider specify the resource context by providing their respective resource-related information. They also update this information when changes occur.

Environment context. The environment context contains information about where the user is located [5] and about its surrounding environment like the current weather or date [21]. This information also includes about the network, which is not within direct control of service user or provider [9,17]. The network that is not under the user's or provider's responsibility can have an immediate impact on the service performance. The environment context depends on the user of the service and is specified by the SLA manager.

Provider context. Provider context covers, among others, information about the provider's current execution load, the duration of its current opened sessions and announced intended length of usage by the application requesting access [9]. All activities performed by the web service and its execution charge have a direct impact on the service's QoS. Increasing or decreasing the computation charge may require changing the SLA. Provider context is specified and updated by the service provider.

Web service context. The Web Service context refers to nonfunctional characteristics of the service. It provides information about possible ranges of execution time, levels of security, expected best reliability, and so on [13,21]. Latency or security are determined by the service's implementation, while metrics like mean availability or reliability are obtained from its past executions. Any changes in the web service context will affect QoS levels, leading to SLA adaptations. The web service context is specified by the service provider.

Provider, web service and some part of the resource categories are related to elements of the provider side and define the level of service that can be offered. User, environment and the other part of the resource categories concern items of the user side and determine the expected level of service. The modifications of all elements of context categories are performed either by the service provider, or by the service user, except for the environment context that can be affected by external events.

Context Illustration. We illustrate in Table 1 the context elements for services from the case study. Both services are offered by the same provider and are

Table 1. Context particularities of MERIS/MGVI and EOLI-SA services

Category	MERIS/MGVI Regional	EOLI-SA
user context	maximize reliability and minimize execution time the execution time must be inferior to 7 hours by day of the selected period	maximize availability
resources context	high performance computing cluster: 120 CPU, 100 terabytes storage capacity, gigabit LAN	high performance computing cluster: 120 CPU, 100 terabytes storage capacity, gigabit LAN
environment context	service user is an human	service user is another web service
provider context	current execution charge of the computing cluster	current execution charge of the computing cluster
web service context	execution time: 4 and 6 hours by day of the selected period reliability: upper than 95% availability: upper than 92%	execution time: inferior to 1 min reliability: upper than 98% availability: upper than 98%

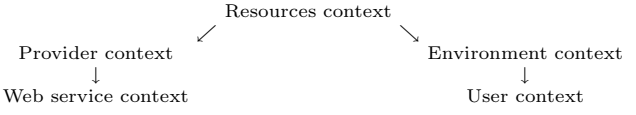
executed on the same computing cluster. Their provider and resources elements are consequently similar.

The MERIS/MGVI service has an important execution time. To prevent failures and potential restart of the execution, the user wishes both to minimize the execution time and maximize the reliability of the web service. Moreover, the user adds a hard constraint on the execution time, stating that it must be inferior to 7 hours by day of the selected period. This constraint prevents an accumulation of unfulfilled requests by the MERIS/MGVI service. The EOLI-SA service has a faster execution, its reliability is not so critical. As it is used by other services to compute data, its availability must be maximized to increase reliability of these other services.

3.2 Context Dependencies

Context dependencies refer to relationships that exist between distinct context elements. For example, QoS properties supported by a service provider can be interrelated [28] e.g.; change in the execution time can affect reliability. These relationships can also occur over elements in different context categories – e.g., the execution charge of computing resources (provider context) affects various quality characteristics of the service (web service context). To highlight such dependencies between elements allows us to propagate failures and performance modifications to all context categories related to the initial variation. Dependencies are defined over context elements with an associated *coefficient*, *direction* and *strength*. The *coefficient* attribute specifies that context elements involved in the dependency are parallel or opposite, meaning that their coefficient is positively or negatively correlated. The *direction* determines, which of the two considered context elements induces the value of the other; a dependency can be directed both ways, meaning that both context elements impact each other. The *strength*, represented by a value between 1 and 10, corresponds to the importance of the influence.

While all context dependencies occurring on the same context category are allowed, between-category dependencies are subject to some restrictions. We specify in Section 3.1 that part of the resources context, the provider context and the web service context are defined by the service provider. The other part of the resources context, the environment context and the user context are delimited by the service user. Dependencies can not involve influence of provider context categories to user context categories and vice versa. Dependencies are also constrained by the attribute direction over different categories. An improvement of the computing resources (resources category) can induce the service quality performance (web service context). Nevertheless, the service quality performance (web service context) has no influence on the computing resources (resources category). The impact direction of dependencies in context categories is given below:



Examples of context dependencies. Context dependencies are specified by the SLA manager to indicate existing interactions between context elements. Table 2 gives examples of context dependencies for MERIS/MGVI and EOLI-SA services.

Table 2. Examples of Context Dependencies

Common dependencies of MERIS/MGVI and EOLI-SA services		
Dep 1		Resources context - Web Service context
	Coefficient:	parallel
	Direction:	→
	Strength:	10
Dependencies of the MERIS/MGVI service		
Dep 2		Execution charge of the computing cluster (provider context) - Execution time (web service context)
	Coefficient:	parallel
	Direction:	→
	Strength:	6
Dep 3		Execution charge of the computing cluster (provider context) - Availability (web service context)
	Coefficient:	opposite
	Direction:	→
	Strength:	8
Dep 4		User bandwidth (environment context) - Transfer Time (user context)
	Coefficient:	parallel
	Direction:	→
	Strength:	5
Dependencies of the EOLI-SA service		
Dep 5		Availability (web service context) - Reliability (web service context)
	Coefficient:	parallel
	Direction:	↔
	Strength:	9
Dep 6		Execution charge of the computing cluster (provider context) - Availability (web service context)
	Coefficient:	opposite
	Direction:	→
	Strength:	2

Dep 1 states the relationship between the resources used by the service provider and web service performance. Indeed, the capability of the web service is immediately linked to the resource used to compute the web service. If the server used to compute MERIS/MGVI is down, all its performance indicators will be affected. The EOLI-SA service is also subject to that dependency. *Dep 2* underlines the impact of the execution charge of the cluster on the execution time needed to execute the service. Increasing the execution charge of the provider decreases resources allocated to the execution of the service and increases its execution time. *Dep 3* states that the increasing of the execution charge of the cluster will decrease the availability of the service: if the cluster charge is full, the MERIS/MGVI service that requires an important resources utilization will not be given a high priority. Consequently, its availability will be reduced. *Dep 4* is about the influence of the user's bandwidth on the user's network capacities. If the bandwidth provided by its Internet Service Provider decreases, the service user will amend its expected total transfer time. The EOLI-SA service is less subject to context variations because it does not consume that much resources. It is subject to the dependency linking its availability to its reliability: *Dep 5*. This dependency is directed both ways meaning that the increasing/decreasing of one of the quality property affectes the other. It is also subject to the *Dep 6* stating that the increasing of the execution charge of the cluster causes a diminution of the availability. This dependency is the same that the one observed for the MERIS/MGVI service but its strength is not so prominent because EOLI-SA does not require a long execution time and is less subject to the provider's utilization.

4 Dynamic SLA Adaptation

We outline here our adaptation process that fits SLA established between service user and provider to context elements variations. Section 4.1 gives a SLA description and presents our SLA management architecture. Section 4.2 details the different steps of our adaptation process.

4.1 Managing Service Level Agreements

Contracts between a service provider and a service user are given by SLAs [11]. An SLA covers the functional side of the service (the provided service corresponds to the requested service in terms of input, output, pre- and postconditions) and concerns also the nonfunctional properties of the service. When users can choose among a set of functionally equivalent web services, QoS considerations become the key criteria for service selection. As a consequence, SLAs must be defined and managed between service users and providers [3]. The contract about non-functional properties is defined for each QoS property through a Service Level Objective (SLO) [22]. SLOs are defined over QoS values and appropriate metrics. Definitions of metrics include the description of their calculation mode and are provided by the party in charge of measurement and aggregation,

i.e.; either the service provider, the service user or a third tier manager [3,29]. An SLA is then a contract between the service user and service provider about a set of SLOs. These SLOs refer to web service and user context elements. Web service context elements are QoS capabilities of the provider while user context elements are quality requirements of the service user. A complete definition of a SLA and its component is available in [23]. We work on the assumption that an initial SLA has already been negotiated between the service user and the service provider with a negotiation process such as the one specified in [29]. We propose in Table 3 examples of SLAs established between providers and users for both services presented in the case study section.

Table 3. Examples of SLAs

SLA established for the MERIS/MGVI service	
SLO 1:	the execution time must be between 5h and 6h by day of the selected period
SLO 2:	the reliability must be superior to 90 %
SLO 3:	the availability must be superior to 80%
SLO 4:	the network time must be inferior to 1'
SLA established for the EOLI-SA service	
SLO 1:	the execution time must be inferior to 1'30"
SLO 2:	the reliability must be superior to 70%
SLO 3:	the availability must be superior to 92%

To manage SLAs and their adaptation, we introduce a third-part service: the *SLA manager*, in charge of the mediation between the service user and the service provider. The adaptation process refers to automatic monitoring, enforcement and optimization of SLAs between the services's user(s) and provider. The SLA manager is also responsible of the assignation of services to users. Our management architecture is illustrated in Figure 5. We dedicate one SLA manager for each existing cluster of web services (i.e., services that offer the same functionality). Gathering of functionally equivalent web services is ensured by means of clusters of web services, that provide several web services inside a unique wrapper, used by the clients as a standard web service [6]. We suppose than an initial SLA has already been negotiated between the service user and the service provider chosen with an adequate selection method [14].

The role of the SLA manager is to continuously check the conformance of the web service to the SLA established between the user and the provider. This monitoring requires a constant verification of SLOs compliance between the service user and the service provider. To achieve this verification, context information about the web service, the provider and the part of the resources information are given by the provider while information related to the user context, its environment and its resources are communicated by the service user. The SLA manager records information about all context elements and builds execution statistics about mean observed latency, reliability or availability. The SLA manager also monitors context dependencies with help of information provided by the user and the provider. With such statistics and information about the execution context and dependencies, the SLA manager is able to check the conformity of SLOs

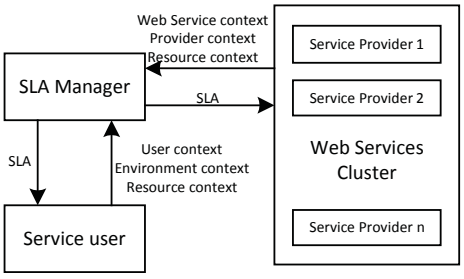


Fig. 5. SLA Management Architecture

established between the service user and the service provider. If some SLOs are breached, the SLA manager processes adaptation mechanisms to adjust the SLA, as explained in Section 4.2.

4.2 Adapting Service Level Agreements

The SLA manager is designed to respond to eventual SLA breaches or QoS variations through different mechanisms of adaptation. *Adaptation* usually refers to the alteration of an application's behavior or interface in response to arbitrary context changes [2]. For web services, the adaptation must consider all context particularities introduced in Section 3.1 as well as existing dependencies over context elements presented in Section 3.2. The aim of such adaptation mechanisms, referred as *SLA adaptation*, is to adjust the initial SLA to context variations reported to the SLA manager. If the initial web service provider is no longer able to perform its task to the quality level requested by the user, the SLA manager proceeds to *select a new provider*. It establishes a new contract between the service user and a service provider selected in the cluster of available services.

The SLA manager process this adaptation through four steps:

1. Modification notification. The SLA adaptation process is driven by the observation of a modification in at least one context category. Such changes are highlighted by information provided by users and providers and statistics made by the SLA manager. The adaptation is initiated differently following the category of the context variation. Provider, web service and some part of the resources context come from the service provider and their changes will modify the service offered, while the user, the environment and the other part of the resources context are defined by the service user and will affect the service level expected.

2. Modification spreading. The second step of the SLA adaptation is the propagation of observed context variation to elements of the same category and to other relevant context categories. Spreading the modification is subject to rules presented in Section 3.2, which define the direction of the allowed dependencies. The impact of the context variations is governed by the coefficient,

direction and strength attributes and reflects changes to all elements of the concerned context categories. Context dependencies allow the SLA manager to propagate the impact of context categories until their influence to related QoS: all context variations are converted to elements used in the SLA contract (i.e., user and web service context).

3. Compatibility checking. Once all dependencies have been propagated, the SLA manager is able to determine the final quality expectations of the user and the web service QoS offered by the service provider. To ensure their compatibility, the SLA manager checks context elements accounted for in the SLA – i.e., the user and web service context. If the web service context presents abilities that meet the expectations of the user context, these are compatible.

4. Adaptation. The last step of the process is the adaptation resulting from compatibility checking. Three different scenarios are possible. (1) The compatibility is present between web service and user context and the initial SLA is still applicable. In this instance, the SLA is preserved between stakeholders. (2) The compatibility is verified between web service and user context but the initial SLA no longer applies. The SLA initiates the set up of a new SLA between the current provider and the service user. To achieve the negotiation between the user and the provider, the manager uses a negotiation process such as the one proposed in [29]. (3) The last possibility occurs while the compatibility between the user and the provider is not verified. The SLA manager then select another service able to meet the quality requirements of the user context in the web services cluster. We do not review here details of selection mechanisms but various existing approaches [7,14,31] can be applied by the SLA manager. The SLA is negotiated between the new provider and the service user by the SLA manager.

Adaptation Illustration. We illustrate here adaptation steps through a particular situation involving services introduced in the case study.

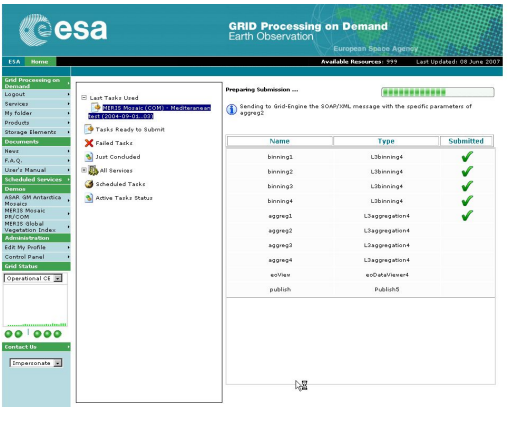


Fig. 6. Current tasks of the provider

The adaptation process described here occurred with an increase of the execution charge in the computing cluster. The execution charge of the computing cluster belongs to the provider context category. The services accessing the computing cluster offered by the ESA are monitored and managed through a particular access interface illustrated in Figure 6. The execution charge can significantly increase with entrance of new requests in the computing cluster. The adaptation mechanisms initiated in response to these new requests will differ with the extent of the increasing. The adaptation process initiated by this increasing is described through its four steps here.

The *first step* is the modification notification. The provider charge is monitored through the application illustrated in Figure 6. With this application, the provider is able to notice the growth of the cluster utilization. The cluster is allowed to work without delays within the execution duration advertised at a fixed level of charge. When the charge moves beyond this level, the provider notifies the SLA manager. We observe the effect produced by two different increases: the first case is an increase of the charge of the computing cluster for 20%; the second involves an increase of 50%.

The *second step* of the adaptation process amounts to spread context modifications. The dependencies are directly related to an increase of the execution charge, i.e., *Dep 2, Dep 3, Dep 6. Dep 2* induces an increase of the execution time and *Dep 3* leads to a decrease of the availability of the MERIS/MGVI service. The *Dep 6* leads to a decrease of the EOLI-SA availability. This decrease enables *Dep 5*, which refers to a decrease of reliability. The effects of an increase of the execution charge on service context of both services are: an increase of the execution time and a decrease of the availability for the MERIS/MGVI service; and a decrease of the availability and the reliability for the EOLI-SA service. The web service contexts of both services resulting from increases of 20% and 50% are illustrated in Table 4.

Table 4. Web Service context of MERIS/MGVI and EOLI-SA services after an increasing of the execution charge

Increasing of the execution charge of 20%	
MERIS/MGVI Regional	EOLI-SA
execution time: 5h and 7h hours by day of the selected period	execution time: inferior to 1 min 10 sec
reliability: upper than 95%	reliability: upper than 97%
availability: upper than 85%	availability: upper than 97%
Increasing of the execution charge of 50%	
MERIS/MGVI Regional	EOLI-SA
execution time: 8 and 10 hours by day of the selected period	execution time: inferior to 1 min 30 sec
reliability: upper than 95%	reliability: upper than 80%
availability: upper than 78%	availability: upper than 80%

The *third step* of the SLA manager's process is the compatibility checking between user requirements and the provider's capabilities. With an increase of 20%, the MERIS/MGVI capabilities still meet user requirements. The EOLI-SA

Table 5. SLAs resulting from the increasing of the execution charge

New SLA established for the MERIS/MGVI service with an increasing of the execution charge of 20%	
SLO 1:	the execution time must be between 6h and 7h by day of the selected period
SLO 2:	the reliability must be superior to 90 %
SLO 3:	the availability must be superior to 80%
SLO 4:	the network time must be inferior to 1'
New SLA established for the EOLI-SA service with an increasing of the execution charge of 50%	
SLO 1:	the execution time must be inferior to 1'30"
SLO 2:	the reliability must be superior to 70%
SLO 3:	the availability must be superior to 80%

is also facing the user expectations with this increase of the execution charge. With an increase of 50%, MERIS/MGVI does not meet the constraint on the maximum allowed execution time, so that the compatibility does not verify. In contrast, the EOLI-SA service is still facing the user requirements and does not break any constraint of the user.

The *fourth step* of the SLA manager is the adaptation. With an increase of 20%, the MERIS/MGVI service is in scenario 2; it is compatible with user requirements but breaches the initial SLA: its execution time is above 6 hours by day of the selected period. The SLA between the user and the provider must be renegotiated. This new SLA is illustrated in Table 5. The EOLI-SA service respects the scenario 1; it is still compatible with user requirements and the initial SLA is still applicable. The initial SLA is preserved between the user and the provider. With an incresing of the execution charge of 50%, the MERIS/MGVI service follows the scenario 3. The service fails to meet the constraint stating that the execution time must be inferior to 7 hours by day of the selected period. The SLA manager selects another service in the services cluster that is able to meet user requirements. The EOLI-SA is in the scenario 2; it is compatible with user expectations but the *SLO 3* of its SLA is breached, the availability is inferior to 92%. A new SLA is negotiated between the service user and the provider, it is illustrated in Table 5.

5 Related Work

Adaptation to failures and SLA violations has received attention [1,8,19,24]. However, the influence of context on SLA adaptation has not been studied in depth. Analyzing the impact of context variations on software behavior is a problem outlined in various other areas such as computer human interaction [5], pervasive computing [18,30] and autonomic systems [2,24]. Context-sensitivity is usually defined as an application software system's ability to sense and analyze context from various sources. It lets application software take different actions adaptively in different contexts [18]. In response to these changes, several adaptation strategies exist [4,12]. Among them, In et al. [8] outline the problem caused by QoS of situation-aware applications. The relationships

between changes of situations and resources required to support the desired level of QoS is not clear. They solve this problem with a situation-aware middleware able to predict all QoS requirements of the applications and to analyze tradeoff relationships among the different QoS requirements. The resource availability may be changed according to dynamically varying situations. Such changes in QoS requirements and QoS constraint violations are identified by their middleware that resolves conflicts by rescheduling resources for supporting high priority missions. In contrast to this model, our proposal relies on an existing definition of dependencies between context elements. Moreover, in the web service area, all resources cannot be modified or rescheduled or are even out of the scope of the service provider or the service user. Our model adapts SLA to context changes and does not intervene on the context elements to comply to QoS requirements. Tosic [26] proposes an alternative to custom-made SLA, the utilization of Web Service Offerings which is supported by an infrastructure (WSOI) and a specific language (WSOL) [25]. Each service is proposed with some classes of service that differ in usage privileges, service priorities, response time guaranteed and verbosity of response information. Their approach cuts off the negotiation problem between the service provider and the service user. However, such predefined classes of service only allow a discrete variation of QoS offered to the service user. Classes of services are predefined, limiting their number and therefore the adaptation possibilities.

6 Conclusions and Future Work

The management of SLAs between an user and a provider in the context of web services is essential to enable autonomy of web service executions. It allows an automatic resolution of conflicts occuring after web service failures or updated expectations of the user. The first advantage of our method is the reliance on the identification of context elements and existing dependencies between these context elements. The context and dependencies allow the SLA manager to anticipate problems. The modifications of context elements are reported to the SLA manager by the provider and the user before the service is executed by the service user. Thus, the SLA manager is able to anticipate and adapt consequently the existing SLA or establish a contract with a new provider. The second advantage of our method is that the SLA manager tries to preserve the existing contract between the service provider and the service user. Long term collaborations between stakeholders are protected from the continuous switching over existing services and new services are selected only when the current provider is not able to meet the user expectations.

Future work consists of the definition of an appropriate language that enables us to integrate context and dependencies elements in existing web services architectures and technologies.

References

1. Bianculli, D., Jurca, R., Binder, W., Ghezzi, C., Faltings, B.: Automated Dynamic Maintenance of Composite Services Based on Service Reputation. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 449–455. Springer, Heidelberg (2007)
2. Bradbury, J.S., Cordy, J.R., Dingel, J., Wermelinger, M.: A Survey of Self Management in Dynamic Software Architecture Specification. In: Proc. ACM SIGSOFT Worksh. Self-healing systems, pp. 28–33 (2004)
3. Cappiello, C., Comuzzi, M., Plebani, P.: On Automated Generation of Web Service Level Agreements. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 264–278. Springer, Heidelberg (2007)
4. Cibrán, M.A., Verheecke, B., Vanderperren, W., Suvée, D., Jonckers, V.: Aspect-oriented Programming for Dynamic Web Service Selection, Integration and Management. World Wide Web Journal 10(3), 211–242 (2007)
5. Dey, A.K., Salber, D., Abowd, G.D.: A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction 16(2-4), 97–166 (2001)
6. Fernandez Vilas, J., Pazos Arias, J., Fernandez Vilas, A.: High Availability with Clusters of Web Services. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, pp. 644–653. Springer, Heidelberg (2004)
7. Herssens, C., Jureta, I.J., Faulkner, S.: Dealing with Quality Tradeoffs during Service Selection. In: ICAC 2008: IEEE Int. Conf. Autonomic Comput. (2008)
8. In, H.P., Kim, C., Yau, S.S.: Q-MAR: An Adaptive QoS Management Model for Situation-Aware Middleware. In: Yang, L.T., Guo, M., Gao, G.R., Jha, N.K. (eds.) EUC 2004. LNCS, vol. 3207, pp. 972–981. Springer, Heidelberg (2004)
9. Julien, C.: Adaptive Preference Specifications for Application Sessions. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 78–89. Springer, Heidelberg (2006)
10. Kaminski, H., Perry, M.: SLA Automated Negotiation Manager for Computing Services. In: CEC/EEE 2006: IEEE Int. Conf. E-Commerce Tech. (2006)
11. Ludwig, H., Keller, A., Dan, A., King, R.P., Franck, R.: Web Service Level Agreement (WSLA) Language Specification. IBM Corporation (2003)
12. Lundesgaard, S.A., Lund, K., Eliassen, F.: Utilising Alternative Application Configurations in Context- and QoS- Aware Mobile Middleware. In: Eliassen, F., Montresor, A. (eds.) DAIS 2006. LNCS, vol. 4025, pp. 228–241. Springer, Heidelberg (2006)
13. Maamar, Z., Mostefaoui, S.K., Yahyaoui, H.: Toward an agent-based and context-oriented approach for Web services composition. IEEE Trans. Knowl. and Data Eng. 17(5), 686–697 (2005)
14. Maximilien, M.E., Singh, M.P.: Toward Autonomic Web Services Trust and Selection. In: ICSOC 2004: Int. Conf. Service Oriented Comput. (2004)
15. Menascé, D.A.: QoS Issues in Web Services. IEEE Internet Computing 6(6), 72–75 (2002)
16. Muldoon, C., OHare, G., Phelan, D., Strahan, R., Collier, R.: Access: An agent architecture for ubiquitous service delivery. In: Klusch, M., Omicini, A., Ossowski, S., Laamanen, H. (eds.) CIA 2003. LNCS, vol. 2782, pp. 1–15. Springer, Heidelberg (2003)
17. Myerson, J.: Use SLAs in a Web Services Context, Part 1: Guarantee your Web Service with a SLA. IBM Research Report (2004), `http://www.ibm.com/developerworks/library/ws-sla/`

18. Nahrstedt, K., Dongyan, X., Wichadakul, D., Baochun, L.: QoS-aware middleware for ubiquitous and heterogeneous environments. Comm. Mag., IEEE 19(11), 140–148 (2001)
19. Netto, M., Bubendorfer, K., Buyya, R.: SLA-Based Advance Reservations with Flexible and Adaptive Time QoS Parameters. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 119–131. Springer, Heidelberg (2007)
20. Papazoglou, M.P., Georgakopoulos, D.: Introduction. Comm. ACM 46(10), 24–28 (2003)
21. Qiu, L., Chang, L., Lin, F., Shi, Z.: Context optimization of AI planning for semantic Web services composition. Service Oriented Comput. and Applications 1(2), 117–128 (2007)
22. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A.P.A., Casati, F.: Automated SLA Monitoring for Web Services. In: Feridun, M., Kropf, P.G., Babin, G. (eds.) DSOM 2002. LNCS, vol. 2506, pp. 28–41. Springer, Heidelberg (2002)
23. Sahai, A., Durante, A., Machiraju, V.: Towards Automated SLA Management for Web Services. Research Report HPL-2001-310 (R.1), Hewlett-Packard Laboratories Palo Alto, July 2002 (2002), `http://www.hpl.hp.com/techreports/2001/HPL-2001-310R1.pdf`
24. Skorin-Kapov, L., Matijasevic, M.: Dynamic QoS Negotiation and Adaptation for Networked Virtual Reality Services. In: WOWMOM 2005: IEEE Int. Symp. World of Wireless Mobile and Multimedia Networks, pp. 344–351 (2005)
25. Tosic, V., Pagurek, B., Patel, K.: WSOL - A Language for the Formal Specification of Classes of Service for Web Services. In: ICWS 2003, IEEE Int. Conf. Web Serv. (2003)
26. Tosic, V.: Service offerings for xml web services and their management applications. PhD thesis (2004)
27. Walsh, A.E. (ed.): UDDI, SOAP, and WSDL: The Web Services Specification Reference Book. Prentice Hall Professional Technical Reference, Englewood Cliffs (2002)
28. Wang, C., Wang, G., Wang, H., Santiago, R.: Quality of Service (QoS) Contract Specification, Establishment, and Monitoring for Service Level Management. J. Object Tech. (2007)
29. Yan, J., Kowalczyk, R., Lin, J., Chhetri, M.B., Goh, S.K., Zhang, J.: Autonomous Service Level Agreement Negotiation for Service Composition Provision. Future Generation Computer Systems 23, 748–759 (2007)
30. Yau, S.S., Karim, F., Wang, Y., Wang, B., Gupta, S.K.S.: Reconfigurable Context-Sensitive Middleware for Pervasive Computing. Pervasive Comput. 1(3), 23–30 (2002)
31. Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Trans. Softw. Eng. 30(5), 311–327 (2004)