A WS-Agreement Based SLA Implementation for the CMAC Platform

Adriano Galati[1], Karim Djemame[1], Martyn Fletcher[2,3], Mark Jessop[2], Michael Weeks[2], and John McAvoy[3]

[1] Distributed Systems and Services Research Group, School of Computing, University of Leeds, E.C. Stoner Building, Woodhouse Lane, LS2 9JT, UK
[2] Advanced Computer Architecture Group, Department of Computer Science, University of York, YO10 5DD, UK
[3] Cybula Ltd. R&D Team, Science Park, York, YO10 5DD, UK

Abstract. The emerging transformation from a product oriented economy to a service oriented economy based on Cloud environments envisions new scenarios where actual QoS (Quality of Service) mechanisms need to be redesigned. In such scenarios new models to negotiate and manage Service Level Agreements (SLAs) are necessary. An SLA is a formal contract which defines acceptable service levels to be provided by the Service Provider to its customers in measurable terms. SLAs are an essential component in building Cloud systems where commitments and assurances are specified, implemented, monitored and possibly negotiable. This is meant to guarantee that consumers' service quality expectations can be achieved. In fact, the level of customer satisfaction is crucial in Cloud environments, making SLAs one of the most important and active research topics. This paper presents an SLA implementation for negotiation, monitoring and renegotiation of agreements for Cloud services based on the CMAC (Condition Monitoring on A Cloud) platform. CMAC offers condition monitoring services in cloud computing environments to detect events on assets as well as data storage services.

Keywords: SLA, WS-Agreement, Requirements, Negotiation.

1 Introduction

Cloud computing [1] is emerging as a new computing paradigm and it is gaining increasing popularity throughout the research community. One aspect of the cloud is the provision of software as a service over the internet, i.e. providing applications (services) hosted remotely. Ideally, these services do not require

Corresponding author: Dr. Adriano Galati, recently moved to Disney Research Zurich, Stampfenbachstrasse 48, 8006 Zurich, Switzerland, email: adriano.galati@disneyresearch.com

end-user knowledge of the physical compute resource they are accessing, nor particular expertise in the use of the service they are accessing. Whilst the cloud offers opportunities for remote monitoring of assets, there are issues regarding resource allocation and access control that must be addressed to make the approach as efficient as possible to maximize revenues. From the service provider perspective, it is impossible to satisfy all customers' requests and a balance mechanism needs to be devised through a negotiation process. Eventually, such a process will end up with a commitment between provider and customer. Such a commitment is a commercial contract that guarantees satisfaction of the QoS requirements of customers according to specific service level agreements. SLAs define the foundation for the expected level of service agreed between the contracting parties. Therefore, they must cover aspects such as availability, performance, cost, security, legal requirements for data-placements, eco-efficiency, and even penalties in the case of violation of the SLA. An SLA is defined as an explicit statement of the expectations and obligations that exist in a business relationship between the user and the service provider. A formalised representation of commitments in the form of an SLA document is required to achieve both automated information collection and SLA evaluation. At any given point in time many SLAs may exist, and each SLA in turn may have numerous objectives to be fulfilled. Therefore, QoS attributes need to be explicitly defined with clear terms and definitions by SLAs which exhibit how service performance is being monitored, and what enforcement mechanisms are in place to ensure SLAs are met [2]. Thus, a user accessing Cloud services on demand and with defined QoS agreements which enables commitments to be fulfilled is a primary requirement. The user can specify an SLA which will guarantee resources, provide job monitoring and record violations if they are detected. The SLA document records agreement provenance allowing for auditing mechanisms after it has terminated. Although the cloud computing research community recognises SLA negotiation as a key aspect of the WS-Agreement specifications, little work has been done to provide insight on how negotiation, and especially automated negotiation, can be realised. In addition, it is difficult to reflect the quality aspects of SLA requirements. In this paper, which follows from our previous work [3], we present our SLA implementation for the management of the negotiation, the monitoring and the renegotiation phase of the agreed terms and requirements for the CMAC (Condition Monitoring on A Cloud) platform [4, 5] which offers a range of analytical software tools designed to detect events on assets and complex systems as well as data storage services. For this purpose, we choose the WSAG4J framework [6, 7] which is an implementation of the WS-Agreement standard [8] from the Open Grid forum (OGF). It provides comprehensive support for common SLA management tasks such as SLA template management, SLA negotiation and creation, and SLA monitoring and accounting. In the rest of this paper we present in Section 2 an exhaustive overview of current literature, in Section 3 we introduce our SLA protocol drawing attention to some aspects related to its design and integration in the CMAC platform. In Section 4 we present all of the requirements we consider for CMAC services, in Section 5 we describe our

SLA implementation, and in Section 6 we provide the lessons we have learned throughout this experience. Finally, Section 7 concludes this paper.

2 Related Work

A Service Level Agreement (SLA) is a contract, between the service provider and the customer which specifies the function performed by the service, the agreed bounds of performance, the obligations on both contractual parties and how deviations are handled [8–10]. Unlike the provision of traditional basic services through SLAs, providing support for real-time applications over service-oriented infrastructures is a complex task that requires enhanced SLA protocols. The Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG) of the Open Grid Forum (OGF) has produced the Web Services Agreement (WS-Agreement) standard [8] to create bilateral agreements. Kubërt et al. [11] analyze different fields where SLAs are used, examine the proposed solutions, and investigate how these can be improved in order to better support the creation of real-time service-oriented architectures. The IRMOS project [12] proposes an SLA framework which introduces a chain of linked SLAs implemented on different layers in order to provide support for the provision of real-time applications. Menychtas et al. [13] present a novel cloud platform, which was developed in the frame of the EU-funded project IRMOS targeting soft real-time applications that have stringent timing and performance requirements. Their platform combines Service Oriented Infrastructures (SOIs) [14] with virtualisation technologies to manage and provision computational, storage and networking resources as well as to communicate with legacy systems such as WiFi locators. Ludwig et al. [15] describe the use of WS-Agreement for Service Level Agreements paving the way for using multiple distributed resources to satisfy a single service request. Battre et al. [16] describe the Web Services Agreement Negotiation protocol proposed by the Open Grid Forum to extend the existing specification. This proposal is the result of combining various research activities that have been conducted to define protocols for negotiating service levels or to supersede the existing "take-it-or-leave-it" protocol. The main characteristics of this proposal are the multi-round negotiation capability, renegotiation capability, and compliance with the original specification. Pichot et al. [17] propose and discuss extensions to the WS-Agreement protocol which support dynamic negotiation and creation of SLAs in an efficient and flexible manner. Some examples of WS-Agreement implementations are WSAG4J [6, 7], Cremona [18] and the SORMA project [19]. WSAG4J (WS-Agreement for Java) framework is a tool developed by Fraunhofer SCAI [6, 7] to create and manage service level agreements (SLAs) in distributed systems. It is an implementation of the OGF WS-Agreement specification [8]. WSAG4J helps designing and implementing SLAs and automates typical SLA management tasks like SLA offer validation, service level monitoring, persistence and accounting. It provides infrastructure components for service providers and consumers supporting the negotiation of SLAs as well as the retrieval of information about negotiated SLAs. A possible extension for WSAG4J

would be the support for SLA template deployment from a tool for designing SLA templates. SLA templates can be created during the design process of a service and the deployment of an SLA template from the design tool would allow a better integration with the SLA infrastructure. Cremona (Creation and Monitoring of Agreements) [18] was developed by IBM using an early version of WS-Agreement. It is a middleware, which supports the negotiation, monitoring, and management of WS-Agreement-based service level agreements. The provided functionality supports both parties involved in the service provisioning and consumption process, i.e. the service provider and the service consumer. SORMA (Self-Organizing ICT Resource Management) [19] implement an Open Grid Market in a comprehensive way by addressing three arguments: the economic model providing an economically sound market structure; the self-organization model, which deals with the interaction between the Grid-application and the market and provides intelligent tools; and the economic middleware model, which builds the bridge between the self-organization and the economic model on the one side and state-of-the-art Grid infrastructure on the other side.

3 SLA Protocol Design

In this section we draw attention to some aspects related to the design and integration of an SLA protocol for the CMAC platform to a point where it can be commercially implemented. Most research up to now provides little insight on how negotiation, and in particular automated negotiation, can be realised. In addition, it is difficult to define the quality aspects of SLA requirements. Here, we have designed an SLA protocol which is integrated in the CMAC platform. For this purpose, the main challenges that we tackle in order to provide QoS guarantees, are mainly three:

- the identification of the QoS properties, i.e. its requirements and terms, and their publication in the SLA template;
- SLA generation and negotiation control based on service requirements and assets available to satisfy customer requests, as well as the decision process to accept, reject, or renegotiate the counteroffer in the negotiation process;
- maximising providers' profit implementing an optimal service and resource allocation policy. Profits are recognised when SLA agreements are honored, generally when workload execution completes on time, otherwise penalties are incurred;

With regard to the first issue, it is not currently tackled by the WS-Agreement specification. We have decided to structure the SLA template distinguishing between *requirements* and *terms*. We assume requirements to describe sufficient conditions required by the service to be executed. More precisely, a service might have need for technical (i.e. a specific operating system, CPU capacity, amount of RAM), syntactical (i.e. defined format of the input data) and ethical requirements (i.e. majority age for the consumer to use the service). Requirements are presented exclusively by the service provider and cannot be negotiable; they are

essential for the fulfillment of the service. Our SLA protocol allows negotiation based on the terms presented by the parties. In this context, the agreed terms are necessary conditions, but not sufficient, to reach an agreement; service requirements must be satisfied anyway. It is necessary to determine all the guarantee terms that will be signed by both parties. For the CMAC services we identify some terms which are QoS parameters like the delivery ability of the provider, the performance of user's workloads, the bounds of guaranteed availability and performance, the measurement and reporting mechanisms, the cost of the service, the terms for renegotiation, and the penalty terms for SLA violation. Our SLA protocol defines ad-hoc SLA template structures for each service on the base of its prerequisites. Our SLA protocol has been designed to allow negotiation only for the guarantee terms. In this respect, the WS-Agreement negotiation protocol decides whether to accept or reject the user's offer, or eventually renegotiate providing a counteroffer, if there are the prerequisites for raising one. The negotiation is quite flexible; it is based on temporal restrictions, resource constraints, previous offers and a maximum number of renegotiations is possible. Once the service requirements and the guarantee terms are met the contract has been stipulated. The negotiation constraints in the negotiation template are used to control the negotiation process. Although, virtualization of resources is a prerequisite for building a successful cloud infrastructure, we do not consider it at this stage. At this point CMAC can start monitoring the service and determining whether the service objectives are achieved or violated. Namely, if the provider has delivered the service within the guaranteed terms. At this early stage we assume only the service provider to be in charge of this process, although each party should be in charge of this task and how fairness can be assured between them is an open issue. The monitoring process helps the service provider to prevent violations of the guarantee terms by renegotiating them.

4 CMAC Requirements

An SLA can define different QoS for each service so as to create ad-hoc service provisions to each service consumer. In this work we target services provided by the CMAC platform [4, 5]. It offers a range of analytical software tools designed to detect events on assets and complex systems as well as data storage services. In particular, in this section, we focus on the *Terms* block of the agreement template. Therefore, we identify requirements and guarantee terms of CMAC services which need to be agreed by service consumers and provider to specify ad-hoc service level agreements before creating a new binding contract. The Table 1 shows the service description terms presented in the implementation of the SLA template for CMAC. Service description terms express a functional description of the provided service. Therefore, such service description terms contain a domain specific description of the service. In particular, the Tables 1(a) and 1(b) list out service constraints and resources derived from the implementation of the CMAC namespace and from the OGF (Open Grid Forum) namespace respectively. CMAC users can select one service among the eeg (elec-

Table 1: Service Description Terms in the CMAC's SLA template.

(a)

CMAC namespace	
Service	eeg ecg storage
Format	csv ndf
Time	Start, end Duration Frequency
Pipeline	Boolean
Browser	Firefox IE
Alerting System Failure	Critical Severe Substantial
Error Handling	Moderate Low Absent
Renegotiation	Boolean
Age of majority	
Communication of violations	email call text post
QoS communication	
Cost	Integer

(b)

OGF namespace	
Operating System (Type, Version)	Linux MacOs Solaris Windows
CPU Architecture (Type)	x86_64 x86_32 ARM sparc PowerPC mips
Physical Memory	Size
Virtual Memory	Size
Disk Space	Size
Bandwidth	Size
Candidate Host	Name
Exclusive Execution	Boolean

troencephalogram), the ecg (electrocardiogram), and the storage of data. The eeg and ecg are both software tools designed for medical signal analysis. Such applications can read data in *cnv* (vector image files primarily associated with DB2 conversion files) and `ndf` format. CMAC allows users to request service provisioning, by allocating time-slots and frequency, between a start-time and end-time. Thus, requested services and resources are available for use during the selected time-slots . Workflows are also supported within the system. This allows users to build a pipeline of individual services and execute them as an orchestrated set of tasks via the workflow execution engine. Processes and output data can be captured via a workflow system, which can orchestrate a pipeline of processing activity. As a matter of course, CMAC archives eeg and ecg output data but they may be further processed for remote visualization on a web browser [4]. Users can decide which connection bandwidth would be suitable for the service performance and the visualization rendering of output data on the web browser. Furthermore, users can choose the alerting level for system errors and service

failures. For that, CMAC can detect anomalies and system failures in signal processing and notify them to service consumers based on the selected alerting level. The error handling level is concerned with recording and communication of failures of the CMAC services. System errors and service failures are notified to CMAC users by means of the preferred communication method, that is, email, call and text. The same communication method is used to notify SLA violations and QoS auditing. CMAC also allows users to upload and make available for use any own software for processing data as long as some requirements such as the operating system and version, CPU architecture, physical memory, virtual memory and disk space can be run on an execution node. Execution nodes can be exclusively allocated or shared among CMAC users. Therefore, users can choose the terms presented in Table 1(b) both for their own and CMAC services. Service consumers can be contacted anytime to renegotiate the agreed terms if the renegotiation flag is set to true. Finally, CMAC users can define maximum cost that they are willing to pay for the service provision and must state they are over eighteen on the SLA template.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:Template wsag:TemplateId="1"
 xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement">
  <wsag:Name>CMAC-1</wsag:Name>
  <wsag:Context>
    <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
    <wsag:TemplateId>1</wsag:TemplateId>
    <wsag:TemplateName>CMAC-TEMPLATE-1</wsag:TemplateName>
  </wsag:Context>
  ...
</wsag:Template>
```

Fig. 1: Name and Context sections of the CMAC's SLA template described by means of the OGF XML Schema.

5 Implementation

The focus on service level rather than on network level enables the definition of SLA and QoS independently from the underlying network technology. Nonetheless, a service should be defined without ambiguity. Therefore, we use the WS-Agreement specification, a standard which defines a language and provides generic support to build common functionalities for advertising the capabilities of service providers, creating agreements based on templates, and for monitoring agreement compliance at runtime. In this context, WSAG4J provides support to design and create highly flexible SLAs using the WS-Agreement language and to validate dynamic agreement offers based on template creation constraints. It also supports common functionality to monitor agreements in a generic way and enables users to build and deploy WS-Agreement based services. WSAG4J allows publishing SLA templates in XML format. The Figure 1 presents the first two sections, *name* and *context*, of an SLA template for the CMAC services which are described by means of the OGF XML schema. We name the agreement described

```
    ...
    <wsag:ServiceDescriptionTerm
     wsag:Name="TIME_CONSTRAINT_SDT" wsag:ServiceName="SAMPLE-SERVICE">
      <wsag4cmact:TimeConstraint xmlns:wsag4cmact=
    "http://schemas.wsag4cmact.org/2012/10/wsag4j-scheduling-extensions">
          <wsag4cmact:StartTime>$STARTTIME</wsag4cmact:StartTime>
          <wsag4cmact:EndTime>$ENDTIME</wsag4cmact:EndTime>
          <wsag4cmact:Duration>$DURATION</wsag4cmact:Duration>
          <wsag4cmact:Frequency>$FREQUENCY</wsag4cmact:Frequency>
      </wsag4cmact:TimeConstraint>
    </wsag:ServiceDescriptionTerm>
...
    <wsag:ServiceDescriptionTerm
     wsag:Name="SERVICE_CONSTRAINT_SDT" wsag:ServiceName="SAMPLE-SERVICE">
      <wsag4cmact:ServiceConstraint xmlns:wsag4cmact=
      "http://www.comp.leeds.ac.uk/2012/10/CMAC-scheduling-extensions">
          $SERVICE
      </wsag4cmact:ServiceConstraint>
    </wsag:ServiceDescriptionTerm>
    <wsag:ServiceDescriptionTerm
     wsag:Name="RENEGOTIATION_CONSTRAINT_SDT"
     wsag:ServiceName="SAMPLE-SERVICE">
      <wsag4cmact:RenegotiationConstraint xmlns:wsag4cmact=
      "http://schemas.wsag4cmact.org/2012/10/CMAC-scheduling-extensions">
          $RENEGOTIATION
      </wsag4cmact:RenegotiationConstraint>
    </wsag:ServiceDescriptionTerm>
    ...
  </wsag:All>
</wsag:Terms>
```

Fig. 2: Resources described by means of the *wsag4cmact* XML Schema in the *terms* block of the CMAC's SLA template.

by the SLA template as *CMAC-1*. In the *context* section we define the party that creates the agreement as "agreement responder", namely the service consumer, and assign him with a unique identifier composed of the template name and id, *CMAC-TEMPLATE-1* and *1* respectively for the sample presented in Figure 1. All of the service description terms in the CMAC SLA template presented in the previous section are exposed in XML format in the *terms* section. Since the WS-Agreement is designed to be domain independent, the content of a service description term can be any valid XML document. In such a section CMAC resources and several constraints are available to users to help specifying requirements and guarantee terms of the provided services. Figure 2 presents CMAC service constraints described by means of *wsag4cmact*, Our XML schema presenting the CMAC service constraints is described by *wsag4cmact*, which is available at http://schemas.wsag4cmact.org/2012/10/wsag4j-scheduling-extensions. Both parties involved in the agreement, i.e. agreement initiator and agreement responder, must understand the domain specific service description. Our XML Schema language is shown in Figure 4. Java interfaces and classes that can be used to access and modify XML instance data have been derived from the XML schema by means of XMLBeans which allows compiling the schema, and generating and jarring Java types. An important requirement for dynamic SLA provisioning is preventing illegal modification of agreement offers. Therefore our

```
<wsag:CreationConstraints>
 ...
 <wsag:ItemConstraint>
  <xs:sequence xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="OperatingSystem" minOccurs="1" maxOccurs="1"
                type="jsdl:OperatingSystem_Type"/>
    <xs:element name="CPUArchitecture" minOccurs="1" maxOccurs="1"
                type="jsdl:CPUArchitecture_Type"/>
    ...
  </xs:sequence>
 </wsag:ItemConstraint>
</wsag:Item>
<wsag:Item wsag:Name="ResourcesSDT_JobDefinition_JobDescription_Resources_
                      CPUArchitecture_CPUArchitectureName">
 <wsag:Location>
   ...
   $this/wsag:AgreementOffer/wsag:Terms/wsag:All/wsag:ServiceDescriptionTerm
   [@wsag:Name='RESOURCE_SDT']/jsdl:JobDefinition/jsdl:JobDescription/
   jsdl:Resources/jsdl:CPUArchitecture/jsdl:CPUArchitectureName
 </wsag:Location>
 <wsag:ItemConstraint>
 <xs:simpleType xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:restriction base="xs:string">
 <xs:enumeration value="sparc"/>
     <xs:enumeration value="powerpc"/>
     <xs:enumeration value="x86_32"/>
     <xs:enumeration value="x86_64"/>
     ...
 </xs:restriction>
 </xs:simpleType>
 </wsag:ItemConstraint>
...
</wsag:CreationConstraints>
```

Fig. 3: Snippets of creation constraints in the CMAC's SLA template.

implementation of the CMAC's agreement template contains several sections of creation constraints which can be used by an agreement responder to define the structure and possible values of valid agreement offers. Only offers that are valid with respect to its template creation constraints are accepted by CMAC. Our creation constraints support CMAC in finding out acceptable values for service descriptions. Besides, they protect the agreement responder from accepting offers that are created in an illegal way. The CMAC agreement template we have designed helps both the agreement initiator and the responder to come to a common understanding of the provided service in the context of an SLA. The agreement offer is valid only if all of the offer items are valid according to the specified item constraints. Figure 3 shows snippets of value and structural constraints described in jsdl of the CMAC SLA templates. The first item refers to the resources defined in the *terms* section of the CMAC SLA template. Here, they must occur once. The second item refers to the CPU architecture name. The location tag points to the `CPUArchitectureName` field which is also defined in the *terms* section. A string type can be assigned to such a variable and CMAC users can choose a CPU architecture out of the ones provided in the list. The third snippet of Figure 3 defines the structural constraints of the agreement offer. They specify which child elements can be part of a certain offer, the order

```
<?xml version="1.0" encoding="UTF-8"?>
  ...
  <xs:complexType name="FrequencyConstraintType">
    <xs:sequence>
      <xs:element name="StartTime" type="xs:dateTime" minOccurs="0" maxOccurs="1" />
      <xs:element name="EndTime" type="xs:dateTime" minOccurs="0" maxOccurs="1" />
      <xs:element name="Duration" type="xs:int" minOccurs="0" maxOccurs="1" />
      <xs:element name="Frequency" type="xs:int" minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="FormatConstraintType">
        <xs:restriction base="xs:string">
                <xs:enumeration value="csv"/>
                <xs:enumeration value="ndf"/>
<xs:enumeration value="plain"/>
        </xs:restriction>
  </xs:simpleType>
  <!-- ecg: electrocardiogram, eeg: elettroencephalogram -->
  <xs:simpleType name="ServiceConstraintType">
<xs:restriction base="xs:string">
<xs:enumeration value="ecg"/>
<xs:enumeration value="eeg"/>
<xs:enumeration value="storage"/>
</xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="AlertingSystemFailureConstraintType">
      <xs:restriction base="xs:string">
                <xs:enumeration value="Critical"/>
                <xs:enumeration value="Severe"/>
            <xs:enumeration value="Substantial"/>
             <xs:enumeration value="Moderate"/>
            <xs:enumeration value="Low"/>
            <xs:enumeration value="Absent"/>
        </xs:restriction>
  </xs:simpleType>
  ...
</xs:schema>
```

Fig. 4: CMAC's XML Schema.

with which such child elements must occur, their cardinality and how they are clustered. This part implies that the *context* and the *terms* sections must be specified in the CMAC SLA template, whereas its *name* section is optional.

6 Lessons Learned

Several questions and problems surfaced during the development of the Service Level Agreement for the CMAC platform. Being CMAC a software platform which offers condition monitoring services to detect events on assets and data storage services in cloud environments, these questions mainly focused on the "how" to implement and "what" requirements need to be taken into account to design an SLA protocol to provide CMAC services. The answers that were found are described here as the lessons learned. Several lessons were learned between the initial analysis phase and the actual implementation of the SLA. During the first phase we identified and analyzed terms and requirements for each of the CMAC services. Therefore, the first lesson is to decide at an early stage what the

provided services are so as to identify all of their requirements and determine how they will be defined in the document structure of the SLA. Besides, the identification of QoS properties, namely, service requirements, guarantee terms and their publication in the SLA template strongly help designing the process to accept, reject or renegotiate an offer and to monitor SLA violations accordingly. For the purpose, we have chosen the WSAG4J framework which helps designing and implementing SLAs and automates typical SLA management tasks like SLA offer validation, service level monitoring, persistence and accounting. In this work we provide a practical lesson on using WSAG4J to define and provide specific services of the CMAC platform. From the best of our knowledge, this is the first employment of WSAG4J for a real-world cloud platform. Therefore, we provide evidence that WSAG4J is a suitable tool for designing SLA agreements. It provides support to design and create highly flexible SLAs using the WS-Agreement language and to validate dynamic agreement offers based on template creation constraints. One more lesson is about the possibility to differentiate between the different SLA components. After investigating service requirements it became clear that some aspects of the service needed more attention than others. Therefore, it would help untangling the components of the provided services and focus on such parts individually, rather than on the services as a whole. Furthermore, the work presented in this paper is part of a common project where researchers and software engineers from both academia and industry are involved. However, some members were not involved in this SLA implementation and would have to work with it as being part of the entire project. This emphasised the need for a specific and readable document based on the WS-Agreement specification and service requirements. An SLA document that can be understood and easily adapted, even by people who have not been involved in the earlier SLA implementation, including descriptions of taken decisions on both document structure and services. A final lesson we learned is that in cloud environments an agreement should not been seen as a rigid contract that cannot be renegotiated and meant to be used only in case of conflicts or agreement violations. It is a document that tries to bring two or more parties into conformity for a common agreement. Therefor commitments on QoS results might not always be enough. A balance between commitments on final results and service performance should be defined by flexible agreements from both service provider and service consumer in order to achieve a fair cooperation. The lessons we have learned provide us with basis references for the development of WS-Agreement based SLAs. Our recommendations provide practitioners with a set of operational SLA concepts. In particular, the identification and definition of requirements of cloud services play a key role in the design of suitable SLA agreements and efficient SLA protocols. By focusing on the gathered requirements and by modelling the SLA process step by step all of the researchers and software engineers involved in this project were able to discuss and contribute to the SLA definition process and the final implementation. Both aspects lead to a well-structured and understandable SLA document, which is the basis for successful service delivery in cloud environments.

7 Conclusions

Currently, WS-Agreement provides little insight on how negotiation, and in particular automated negotiation, can be realized. In addition, it is difficult to define the quality aspects of SLA requirements. This paper presents an SLA protocol designed to guide the negotiation, the monitoring and the renegotiation phase of the agreed terms to maximize revenues in the CMAC platform. Besides, we also provide a clear distinction between quality aspects of SLA requirements. In this work we integrate our SLA protocol into the CMAC platform which offers a range of analytical software tools designed to detect events on assets and complex systems as well as data storage services. For this purpose, we choose the WSAG4J framework which is a tool to create and manage service level agreements in distributed systems. In this work, we have developed the first SLA protocol for the CMAC platform with the intention to maximize revenues by designing an appropriate resource allocation process based on time restrictions and related service parameters agreed during the negotiation phase and possibly modified by means of renegotiations. The lessons learned provide us with basis references for the development of WS-Agreement based SLAs. Our recommendations provide practitioners with a set of operational SLA concepts. In particular, we draw attention to the identification and definition of requirements of cloud services as fundamental for the design of suitable SLA agreements and efficient SLA protocols. In future work we would like to implement and evaluate the renegotiation protocol we have designed in this work. We would also like to consider varying expiration times based on historical records to handle the duration of the negotiation phase so as to enhance its flexibility. Besides, we would also like to provide service consumers with estimates on previous service executions along with service performance measurement methods, measurement periods and data analysis reports, which will help choosing suitable service requirements. Moreover, we are examining the possibility of integrating the SLA protocol with service pricing and discounting policies to apply when SLA commitments are not satisfied.

Acknowledgment

We would like to thank Wolfgang Ziegler, Oliver Wäldrich and Hassan Rasheed from the Fraunhofer Institute SCAI for their valuable advices. This research is partly funded by the University of Leeds through a Knowledge Transfer Secondement grant, and the European Union within the 7th Framework Programme under contract ICT-257115 - OPTIMIS.

References

1. R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg and I. Brandic, *Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility*, Future Gener. Comput. Syst. 25, 6, 599-616 (2009).

2. E. Keller and H. Ludwig, *The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services*, Journal of Network and Systems Management, Vol. 11, pp. 57-81 (2003).
3. A. Galati, K. Djemame, M. Fletcher, M. Jessop, M. Weeks, S. Hickinbotham, J. McAvoy, *Designing an SLA Protocol with Renegotiation to Maximize Revenues for the CMAC Platform*, at the Cloud-enabled Business Process Management (CeBPM 2012), Paphos, Cyprus (2012).
4. S. Hickinbotham, J. Austin and J. McAvoy, *Interactive Graphics on Large Datasets Drives Remote Condition Monitoring on a Cloud*, In the Proceedings of the Open Access Journal of Physics: Conference Series, Part of CMAC project, for Cybula Ltd. and the University of York, COMADEM2012, 18-20 (2012).
5. B. Liang, S. Hickinbotham, J. Mcavoy and J. Austin, *Condition Monitoring Under the Cloud*, In the Proceedings of the Digital Research 2012, Oxford (UK), (2012).
6. D. Battré, M. Hovestadt and O. Wäldrich, *Grids and Service-Oriented Architectures for Service Level Agreements*, Springer, pp. 23-34 (2010).
7. O. Wäldrich and W. Ziegler, *WSAG4J - Web Services Agreement for Java*, [online] Available: https://packcs-e0.scai.fraunhofer.de/wsag4j
8. A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Kakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, *Web Services Agreement Specification (WS-Agreement)*, GRAAP-WG, OGF recommendation, [online] Available: http://www.ogf.org/documents/GFD.192.pdf
9. OGF GFD.120: *Open Grid Services Architecture - Glossary of Terms*, J. Treadwell (2007).
10. *SLA Management Handbook: Volume 2 Concepts and Principles*, Release 2.5, Tele-Management Forum, GB 917-2(2005).
11. R. Kubërt, G. Gallizo, T. Polychniatis, T. Varvarigou, E. Oliveros, S.C. Phillips and K. Oberle, *Service Level Agreements for Real-time Service-Oriented Infrastructures*, Chapter 8 in Achieving Real-Time in Distributed Computing book: From Grids to Clouds. , IGI Global Books, Information Science Pub, (2011).
12. *IRMOS Project*, Available [online]: http://www.irmosproject.eu
13. A. Menychtas, D. Kyriazis, S. Gogouvitis, K. Oberle, T. Voith, G. Gallizo, S. Berger, E. Oliveros and M. Boniface, *A Cloud Platform for Real-time Interactive Applications*, 1st International Conference on Cloud Computing and Service Science (CLOSER 2011), Noordwijkerhout, The Netherlands, (2011).
14. T. Erl, *Service-oriented Architecture: Concepts, Technology, and Design*, Upper Saddle River: Prentice Hall PTR, ISBN 0-13-185858-0, (2005).
15. H. Ludwig, T. Nakata, O. Wäldrich and W. Ziegler Coregrid Tr, *Reliable Orchestration of Resources using WS-Agreement*, High Performance Computing and Communications Lecture Notes in Computer Science Volume 4208, pp 753-762,(20060.
16. D. Battré, F.M.T. Brazier, K.P. Clark, M.A. Oey, A. Papaspyrou, O. Wäldrich, P. Wieder and W. Ziegler, *A Proposal for WS-Agreement Negotiation*, In Proc. of the 11th IEEE/ACM Int. Conference on Grid Computing, pp. 233-241, (2010).
17. A. Pichot, P. Wieder, O. Wäldrich and W. Ziegler, *Dynamic SLA-negotiation based on WS-Agreement*, CoreGRID TR-0082, Technical Report. (2007)
18. H. Ludwig, A. Dan, R. Kearney, *Cremona: An architecture and library for creation and monitoring of WSAgreements*, Proceedings of the International Conference on Service Oriented Computing (ICSOC172004), 65-74, ACM: New York (2004).
19. D. Neumann, J. Stoesser, A. Anandasivam, N. Borissov, *SORMA-Building an open market for grid resource allocation*, Grid Economics and Business Models (Lecture Notes in Computer Science, vol. 4685). Springer 194-200; Berlin (2007).