2009 33rd Annual IEEE International Computer Software and Applications Conference

# Towards Self-manageable Cloud Services

Ivona Brandic
*Distributed Systems Group*
*Institute of Information Systems*
*Vienna University of Technology*
*Vienna, Austria*
*ivona@infosys.tuwien.ac.at*

*Abstract*—**Cloud computing represents a promising computing paradigm, where computational power is provided as a utility. An important characteristic of Cloud computing, other than in similar paradigms like Grid or HPC computing, is the provision of non-functional guarantees to users. Thereby, applications can be executed considering predefined execution time, price, security or privacy standards, which are guaranteed in real time in form of Service Level Agreements (SLAs). However, due to changing components, workload, external conditions, hardware, and software failures, established SLAs may be violated. Thus, frequent user interactions with the system, which are usually necessary in case of failures, might turn out to be an obstacle for the success of Cloud computing. In this paper we discuss self-manageable Cloud services. In case of failures, environmental changes, and similar, services manage themselves automatically following the principles of autonomic computing. Based on the life cycle of a self-manageable Cloud service we derive a resource submission taxonomy. Furthermore, we present an architecture for the implementation of self-manageable Cloud services. Finally, we discuss the application of autonomic computing to Cloud services based on service mediation and negotiation bootstrapping case study.**

*Keywords*-**Cloud Computing; Autonomic Computing; Service Self-management; Web Services;**

## I. Introduction

The Cloud computing paradigm represents an emerging solution for the provision of computational resources as services [6]. Other than in similar paradigms like Grid or HPC computing, computational resources are provided as services with compliance to pre-defined functional and non-functional requirements. Non-functional requirements consider for example application execution time, price, data privacy or applied security standards. On-demand resource discovery and real-time Service Level Agreement (SLA) negotiation with potential partners represent fundamental characteristics of Cloud computing [6]. As already identified contemporary Cloud computing solutions offer computational power towards Web 2.0 style applications [21] with frequent user interactions. Only recently researchers have begun to address the needs of enterprise solutions, such as support for infrastructure-level SLAs [20].

Currently, large body of work is done in various Cloud computing areas, as for example on automatic service deployment using virtualization [10], on storage Clouds for content delivery networks [2], on application of Cloud computing for scientific workflows [16] or on evaluation of opportunities and challenges of Cloud computing as a service [15]. However, very little attention has been paid to the appropriate management of a single Cloud service. Once established, SLAs should not be violated even in case of system failures, changed environmental conditions or unpredictable events. Thus, self-management of a service minimizing user-interactions with the system represents a challenging research issue.

In this paper we present novel concepts for self-manageable Cloud services. Based on the principles of autonomic computing we discuss the life cycle of self-manageable Cloud services based on the following phases: meta negotiation, negotiation, service execution, and post-processing. Following the derived resource taxonomy, which considers major resource submission approaches i.e., simple jobs, workflows, and arbitrary resources, we present an architecture for the self-management of Cloud services. We design service interfaces necessary for the proper interaction with the autonomic manager component and exemplify the self-management process. Furthermore, we discuss the application of the principles of autonomic computing to Cloud services based on negotiation bootstrapping (NB) and service mediation (SM). Based on NB and SM even heterogeneous service providers and consumers e.g., partners supporting different negotiation protocols, may negotiate with each other and generate SLA agreements. Using the principles of SM users can map between inconsistent SLA templates and negotiate with partners even in case of inconsistent SLA templates.

The main contributions of this paper are: (i) discussion on the life cycle of a self-manageable Cloud service; (ii) definition of the resource taxonomy for self-manageable Cloud services; (iii) discussion on the architecture for the self-manageable Cloud services; and (iv) demonstration of the application of autonomic computing to self-manageable Cloud services based on a negotiation bootstrapping and service mediation case study.

The rest of this paper is organized as follows: Section II presents the related work. In Section III we discuss the principles of the autonomic computing. Based on that principles we derive the life cycle of self-manageable Cloud services and discuss the resource submission taxonomy. Furthermore, we present the architecture for self-manageable Cloud services. In Section IV we discuss a case study on service mediation and negotiation bootstrapping. Section V presents our conclusions and describes the future work.

## II. Related Work

Currently, related work on self-manageable Cloud services can be classified in following categories: (i) novel solutions for Cloud computing approach [2], [20], [18]; (ii) SLAs and negotiation in related areas as Grid computing [7], [1], [19]; (iii) economy of Cloud computing [6], [8]; (iv) application of autonomic computing to related areas [13], [14].

Rochwerger et al. identify inherently limited scalability of single-provider clouds, lack of interoperability among cloud providers, and no built-in Business Service Management (BMS) support as shortcomings of contemporary clouds [20]. They present an architecture for federation of disparate data centers based on the principles of BMS. Nurmi et al. discuss Eucalyptus, an open-source software framework for Cloud computing, that implements the Infrastructure as a Service (IaaS) principle [18].

Work presented in [1] extends the service abstraction in the Open Grid Services Architecture (OGSA) for Quality of Service (QoS) properties focusing on the application layer. Thereby, a given service may indicate the QoS properties it can offer or it may search for other services based on specified QoS properties. Work presented in [19] discusses incorporation of SLA-based resource brokering into existing Grid systems. However, both approaches do not consider heterogeneity of Cloud computing platforms. Work presented in [6] discusses the impact of Cloud computing for the provision of computing power as a 5th utility. Deelman et al. analyze the cost of doing science on the Cloud based on the Montage example [8]. Montage is an image mosaicking application, which requires large scale computational resources.

Based on the defined workflows adaptations as MAPE[1] decision making [14], Lee et al. discuss the application of autonomic computing to the adaptive management of Grid workflows [13]. Thereby, a scheduling algorithm, that allocates resources based on runtime performance, is discussed and the proposed approach is evaluated using a real world Grid workflow system.

Although, several approaches deal with the application of autonomic computing to Grids and Grid workflows, to the best of our knowledge none of the presented approaches deals with the self-management of Cloud services.

## III. Self-management for Cloud Services

In this section we model an architecture for self-manageable Cloud services. In Section III-A we discuss the principles of autonomic systems. In Section III-B we present the life cycle of a self-manageable Cloud service based on the principles of autonomic computing. In Section III-C we discuss a taxonomy on resources, which can be submitted to a service. Based on that taxonomy we model an architecture for service self-management (Section III-D).

[1]Monitoring, Analysis, Planning, Execution

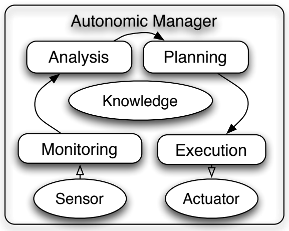

Figure 1. General Architecture of an Autonomic System

### A. Autonomic Systems

Autonomic systems require high-level guidance from humans and decide, which steps need to be done to keep the system stable [12]. Such systems constantly adapt themselves to changing environmental conditions. Similar to biological systems (e.g., human body) autonomic systems maintain their state and adjust operations considering changing components, workload, external conditions, hardware, and software failures. Usually, autonomic systems comprise one or more managed elements e.g., QoS elements.

An important characteristic of an autonomic system is an intelligent closed loop of control. As shown in Figure 1 the autonomic manager manages the element's state and behavior. Thereby, sensed changes to managed resources result in the invocation of the appropriate set of actions required to maintain some desired system state. Typically control loops are implemented as MAPE (monitoring, analysis, planning, and execution) functions [12]. The *monitor* collects state information and prepares it for the *analysis*. If deviations to the desired state are discovered during the analysis, the *planner* elaborates change plans, which are passed to the *executor*.

### B. Life cycle of a Self-manageable Cloud Service

Figure 2 depicts the life cycle of a self-manageable Cloud service. As depicted in Figure 2 we identified three phases: *meta negotiation*, *negotiation*, and *service execution*:

- **Meta negotiation.** *Meta negotiation* is necessary in order to select services a service provider may negotiate with. During the *meta negotiation* phase supported negotiation protocols, supported SLA specification languages (e.g., WSLA [23], WS-Agreement), and similar issues are negotiated. However, during this phase self-management abilities are necessary, if for example different negotiation protocols are available

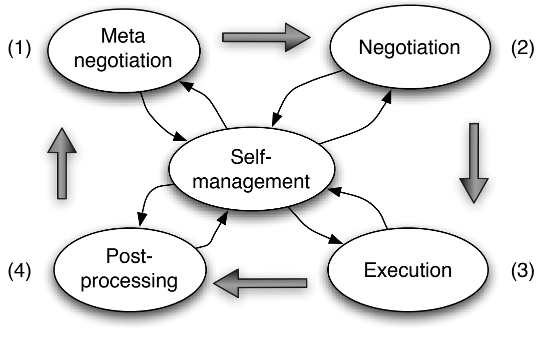

Figure 2. Life cycle of a Self-manageable Service

on the consumer's and provider's side. Thus, the self-management component bootstraps between both parties. The application of the self-management process to meta negotiation is exemplified in Section IV-A.

- **Negotiation**. After the meta negotiation is concluded, the *negotiation* phase may start. During the negotiation phase the parties agree on specific terms of contract, as for example execution time or price. However, even during that phase self-management component facilitates the negotiation process. An example for self-management during the negotiation phase is the service mediation between non matching SLA templates as demonstrated in Section IV-B.
- **Execution.** After the negotiation phase, *service execution* may start. The service execution phase includes job submission, job monitoring, upload of input data, download of output data, and similar actions. An example for self-management during the execution phase is automatic job migration, if the agreed job execution time seems to be violated.
- **Post-processing.** During the *post-processing* phase the consumed resources are released after the successful or not successful service execution. An examples for resource self-management during the post-processing phase is the release of resources, if jobs complete earlier than estimated. Reasons for an earlier job completion could be successful job terminations earlier than estimated or job breakups in case of inability to complete jobs.

In the following we discuss, which resources can be submitted for execution to a self-manageable Cloud service.

### C. Resource Submission Taxonomy

During the negotiation phase the participating parties may negotiate on functional and non-functional requirements. Thus, the service user may submit different kinds of resources[2]. An example are native jobs, which are submitted to the service provider, if functional requirements of the service consumer match with those of the service provider. As specified in Table I and based on the provider's system and user requirements, users may submit native jobs, native workflows, and arbitrary resources. In the following we present the resource submission taxonomy:

| Resource | Description | Example |
|---|---|---|
| **Job** | Job Description Language (JDL) | JSDL |
| **Workflow** | Workflow Language (+ JDL) | QoWL + JSDL |
| **Arbitrary** | Service Specification Language | Service Manifest |

Table I
Resource Submission Taxonomy

- **Job.** If possible consumers may run their native jobs on the provider's platforms. This is of course only the case, if the required consumer's platform matches the provided platform. Whether a native job can be executed or not is found out during the negotiation phase. Native jobs are specified using job description languages, as for example the Job Submission Description Language (JSDL) format [17]. Using JSDL the job name and similar parameters can be specified, as for example job description, resource requirements that computers must have to be eligible for scheduling, such as total RAM available, total swap available, number of CPUs, OS, etc.
- **Workflow.** Similar to simple native jobs complex workflows can be specified and executed. Additional to the workflow specification a job description file might be submitted, as for example QoWL workflows [3] and JSDL job description files. However, in some systems only a workflow specification file may be sufficient, if the functional requirements are specified within the workflow specification file (e.g. Pegasus workflows [8]). Initial work on self-manageable workflows is presented in [13].
- **Arbitrary.** In case of arbitrary resources, e.g. if resources should be executed on arbitrary OS systems using arbitrary middleware, the service consumer has to specify the service description using a service specification language. An example for service specification languages is a Service Manifest as proposed by the Reservoir project [20]. Using a manifest users describe a master image by means of a self-contained software stack (OS, middleware, applications, data, and configuration), that fully captures the functionality of the component type. Moreover, the manifest contains information and rules necessary to automatically create, from a single parameterized master, service instances. The provision of arbitrary resources is explained in [11]. We propose an SLA based resource virtualization approach for on-demand service provision. Based on

[2]Please note: In this section resources address job submission types like jobs, workflows or arbitrary services.

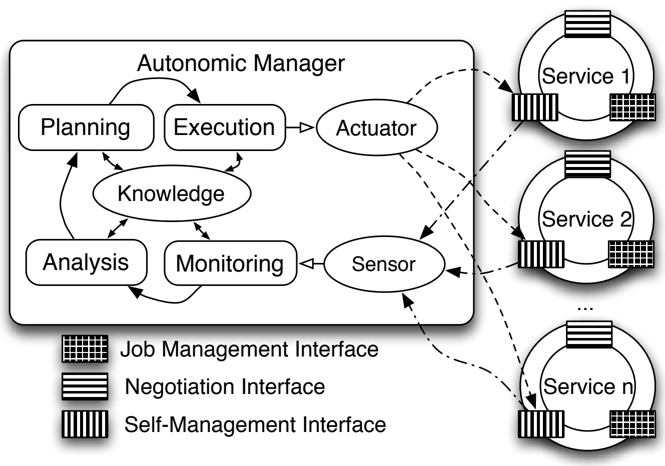

Figure 3. Architecture for Self-manageable Cloud Service

that approach SLAs are negotiated in a hierarchic way using meta negotiations, meta broker, broker, and finally automatic service virtualization and deployment.

Details on self-management during service execution are explained in the following.

### D. Architecture for Self-manageable Cloud Services

Figure 3 depicts the architecture for self-manageable Cloud services. After meta negotiation and negotiation, services are deployed on demand as described in [11]. All deployed services implement three interfaces for *job management*, *autonomic management*, and *negotiation* as described next.

- **Negotiation.** Negotiation interfaces are known before the service deployment process. However, due to autonomic self-management it might be necessary to re-negotiate established SLAs during service execution. The negotiation interface implements negotiation protocols, SLA specification languages, and security standards as stated in the meta-negotiation document (see Figure 6).
- **Job Management.** Job management interfaces are necessary for the manipulation of the jobs, as for example the upload of input data, download of output data, start of the job or similar. Job management interfaces are provided by the service user described in the Service Manifest and are automatically utilized during the service deployment process.
- **Autonomic Management.** As shown in Figure 3 in this section we focus on the autonomic management interface. This interface is implemented by each arbitrary Cloud service and specifies operations for sensing changes of the desired state and for reacting to those changes. Sensors are activated using the notification approach implemented using the WS-Notification standard [22]. Thus, sensors may subscribe to a specific topic e.g., violation of execution time. Based on the measured values as demonstrated in [5] notifications are obtained, if the execution time is violated or seams to be violated very soon. After the activation of the control loop the service actuator reacts and invokes proper operations e.g., the migration of resources.

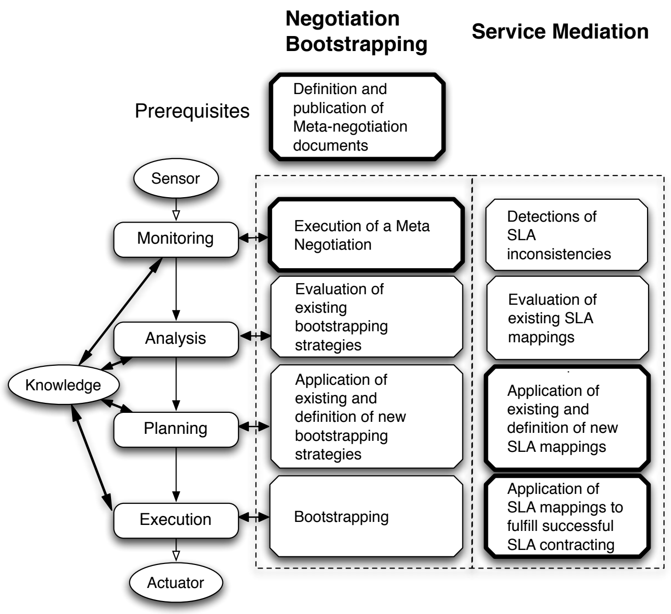
Negotiation Bootstrapping
Service Mediation
Prerequisites
Definition and publication of Meta-negotiation documents
Sensor
Monitoring
Execution of a Meta Negotiation
Detections of SLA inconsistencies
Analysis
Evaluation of existing bootstrapping strategies
Evaluation of existing SLA mappings
Knowledge
Planning
Application of existing and definition of new bootstrapping strategies
Application of existing and definition of new SLA mappings
Execution
Bootstrapping
Application of SLA mappings to fulfill successful SLA contracting
Actuator

Figure 4. Negotiation Bootstrapping and Service Mediation as Part of the Autonomic Process

In the following we demonstrate the application of the autonomic computing to Cloud services based on negotiation bootstrapping and service mediation.

## IV. Application of Autonomic Computing to Cloud Services

In this section we demonstrate, how the principles of autonomic computing can be applied to the self-management of Cloud services. In Section IV-A we present a case study for an autonomic system based on negotiation bootstrapping, whereas in Section IV-B we present a case study for an autonomic system based on service mediation. In Section IV-C we discuss solutions for the realization of negotiation bootstrapping.

### A. Negotiation Bootstrapping Case Study

Before using the service, the service consumer and the service provider have to establish an electronic contract defining the terms of use [3], [1]. Thus, they have to negotiate the detailed terms of contract e.g., the execution time of the service. However, each service provides a unique negotiation protocol, often expressed using different languages representing an obstacle within the SOA architecture and especially in emerging Cloud computing infrastructures. We propose novel concepts for automatic bootstrapping between different protocols and contract formats increasing the number of services a consumer may negotiate with. Consequently, the full potential of publicly available services could be exploited.

Figure 4 depicts how the principles of autonomic computing can be applied to negotiation bootstrapping and service mediation. The management is done through following steps: As a prerequisite of the negotiation bootstrapping users have to specify a meta negotiation (MN) document describing the requirements of a negotiation, as for example required negotiation protocols, required security infrastructure, provided document specification languages, etc. The autonomic management phases are described in the following:

- **Monitoring.** During the monitoring phase all candidate services are selected, where negotiation is possible or bootstrapping is required.
- **Analysis.** During the analysis phase the existing knowledge base is queried and potential bootstrapping strategies are found.
- **Planning.** In case of missing bootstrapping strategies users can define new strategies in a semi-automatic way.
- **Execution.** Finally, during the execution phase the negotiation is started by utilizing appropriate bootstrapping strategies.

## B. Service Mediation Case Study

In this section we apply the principles of autonomic computing to service mediation as described next:

- **Monitoring.** During service negotiation and based on monitoring information inconsistencies in SLA templates may be discovered. Inconsistencies may include for example different indicators for the same term of contract e.g., the service price, which may be defined as the *usage price* at the consumer's side and as the *service price* at the provider's side.
- **Analysis.** During the analysis phase existing SLA mappings are analyzed e.g., existing and similar SLA mapping are queried.
- **Planning.** During the planning phase new SLA mappings can be defined, if existing SLAs cannot be applied.
- **Execution.** Finally, during the execution phase the newly defined SLA mappings can be applied and the negotiation between heterogeneous consumers and providers may start.

An implementation of the SLA mapping approach is demonstrated in [5].

As indicated with bold borders in Figure 4, we already implemented solutions for the definition and accomplishment of meta-negotiations and for the specification and applications of SLA mappings in our previous work [3], [5], [4].

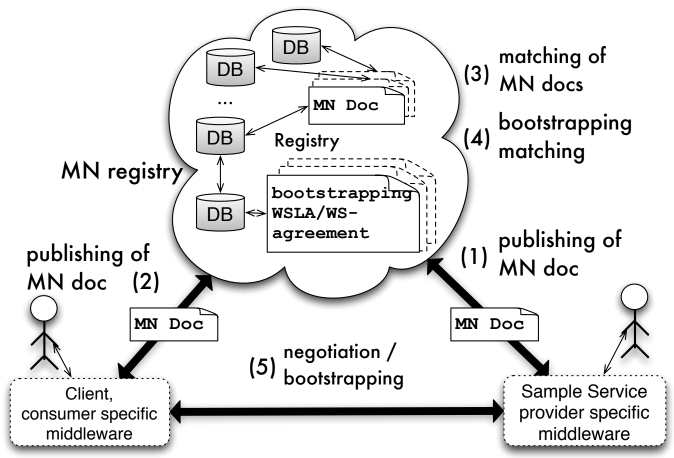
DB
DB
...
DB
DB
MN Doc
Registry
bootstrapping WSLA/WS-agreement
MN registry
(3) matching of MN docs
(4) bootstrapping matching
publishing of MN doc (2)
(1) publishing of MN doc
MN Doc
MN Doc
(5) negotiation / bootstrapping
Client, consumer specific middleware
Sample Service provider specific middleware

Figure 5. Negotiation Bootstrapping Approach

```
<meta-negotiation ...>
  ...
 <pre-requisite>
  <security>
   <authentication value="GSI" location="uri"/>
  </security>
  <negotiation-terms>
   <negotiation-term name="beginTime"/>
   <negotiation-term name="endTime"/>
    ...
  </negotiation-terms>
</pre-requisite>
<negotiation>
 <document name="WSLA" value="uri" .../>
 <protocol name="alternateOffers"
   schema="uri" location="uri" .../>
</negotiation>
<agreement>
 <confirmation name="arbitrationService" value="uri"/>
</agreement>
</meta-negotiation>
```

Figure 6. Example Meta Negotiation Document

## C. Negotiation Bootstrapping Realization

Figure 5 depicts the bootstrapping approach. A service provider publishes descriptions and conditions of supported negotiation protocols into the registry (step 1). Thereafter, service consumers perform lookups on the registry database by submitting their own documents describing the negotiations that they are looking for (step 2). The registry discovers service providers, who support the negotiation processes that a consumer is interested in and returns the documents published by the service providers (step 3). If the consumer and provider understand different negotiation protocols e.g., WSLA and WS-Agreement, the registry checks, whether a bootstrapping strategy for WSLA / WS-Agreement is specified or not (step 4). Finally, after an appropriate service provider and a negotiation protocol are selected by a consumer using his/her private selection strategy, negotiations between them may start according to the conditions specified in the provider's document (step 5). At the same time bootstrapping strategies can be applied, if necessary.

Bootstrapping can be done via wrappers, which can be downloaded from the registry. As depicted in Figure 5 it is possible to host the registry using a cloud of databases hosted on a service provider, such as Google App Engine or Amazon S3 [9], [21].

A sample meta negotiation (MN) document is depicted in Figure 6. The conditions having to be satisfied before negotiation are defined within the `<pre-requisite>` element (see lines 3–12). Pre-requisites include the *security credentials* and the *negotiation terms*. Details about the negotiation process are defined within the `<negotiation>` element. In Figure 6 *WSLA* is specified as the supported document language. Additional attributes specify the URI to the API or WSDL for the documents and their versions supported by the consumer. In Figure 6 *AlternateOffers* is specified as the supported negotiation protocol. However, if there are bootstrapping strategies defined e.g., between alternate offers and reverse english auction, the service provider could even negotiate with services, which only understand the reverse english auction negotiation protocol. The `<agreement>` element may be verified by a third party organization or may be lodged with another institution, who will also arbitrate in case of a dispute as specified by the `<agreement>` element.

## V. Conclusion and Future Work

In this paper we presented first achievements towards self-manageable Cloud services. Considering the changing components, workload, external conditions, hardware, and software failures services manage themselves automatically to keep the system's state stable. Based on a life cycle of a self-manageable Cloud service we derived a resource submission taxonomy considering simple jobs, workflows, and arbitrary Cloud services. Furthermore, we presented an architecture for the implementation of self-manageable Cloud services including interfaces for job management, negotiation, and autonomic self-management. Finally, we discussed the application of autonomic computing to Cloud services based on service mediation and negotiation bootstrapping case studies.

In the future we plan to identify monitoring elements correlated with the corresponding sensing elements and to derive control loops of those elements.

## Acknowledgment

The work described in this paper was partially supported by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII). The author would like to thank Rizos Sakellariou for sharing thoughts, for the fruitful discussions, and valuable comments on the early versions of the paper. Finally, the author would like to thank anonymous reviewers for their helpful and constructive comments, which have helped the revision of the paper enormously.

## References

[1] R. J. Al-Ali, O. F. Rana, D. W. Walker, S. Jha, and S. Sohail. *G-qosm: Grid service discovery using qos properties*. Computing and Informatics, 21:363–382, 2002.

[2] J. Broberg, Z. Tari: MetaCDN: *Harnessing Storage Clouds for High Performance Content Delivery*. the 6th International Conference on Service-Oriented Computing (ICSOC) 2008, Sydney, Australia, December 1-5, 2008.

[3] I. Brandic, D. Music, S. Dustdar, S. Venugopal, and R. Buyya. *Advanced QoS Methods for Grid Workflows Based on Meta-Negotiations and SLA-Mappings*. The 3rd Workshop on Workflows in Support of Large-Scale Science. In conjunction with Supercomputing 2008, Austin, TX, USA, November 17, 2008.

[4] I. Brandic, D. Music, S. Dustdar. Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services. Grids meet Autonomic Computing Workshop 2009 - GMAC09. In conjunction with the 6th International Conference on Autonomic Computing and Communications Barcelona, Spain, June 15-19, 2009.

[5] I. Brandic, D. Music, P. Leitner, S. Dustdar.*VieSLAF Framework: Increasing the Versatility of Grid QoS Models by Applying Semi-automatic SLA-Mappings*. Vienna University of Technology, Technical Report, TUV-184-2009-02.pdf, 2008.

[6] R. Buyya, Ch. 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. Technical Report, GRIDS-TR-2008-13, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Sept. 27, 2008.

[7] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. *Web services on demand: WSLA-driven automated management*. IBM Systems Journal, 43(1), 2004.

[8] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good. *The Cost of Doing Science on the Cloud: The Montage Example.* Proceeding of Super Computing 2008, Austin, Texas, November 2008.

[9] Google App Engine, http://code.google.com/appengine

[10] G. Kecskemeti, P. Kacsuk, G. Terstyanszky, T. Kiss, T. Delaitre. *Automatic Service Deployment Using Virtualisation*. 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), Toulouse, France, 13-15 February 2008.

[11] A. Kertesz, G. Kecskemeti, I. Brandic. An SLA-based Resource Virtualization Approach for On-demand Service Provision. VTDC 2009 - The 3rd International Workshop on Virtualization Technologies in Distributed Computing. In conjunction with the 6th International Conference on Autonomic Computing and Communications Barcelona, Spain, June 15-19, 2009.

[12] J.O. Kephart, D.M. Chess, *The vision of autonomic computing.* Computer, 36:(1) pp. 41-50, Jan 2003.

[13] K. Lee, N. W. Paton, R. Sakellariou, E. Deelman, A. A. A. Fernandes, G. Mehta. *Adaptive Workflow Processing and Execution in Pegasus*. 3rd International Workshop on Workflow Management and Applications in Grid Environments (WaGe08), in Proceedings of the Third International Conference on Grid and Pervasive Computing Symposia/Workshops, Pages 99-106, ISBN 978-0-7695-3177-9, May 25-28 2008, Kunming, China.

[14] K. Lee, R. Sakellariou, N. W. Paton, A. A. A. Fernandes. Workflow *Adaptation as an Autonomic Computing Problem*. Proceedings of the 2nd workshop on Workflows in support of large-scale science (WORKS'07), 2007, Monterey, California, USA (in conjunction with HPDC 2007), pp. 29-34.

[15] G. Lin, G. Dasmalchi, J. Zhu. *Cloud Computing and IT as a Service: Opportunities and Challenges*. IEEE International Conference on Web Services (ICWS08), Beijing China, 23-26 Sept. 2008.

[16] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, J. Good. *On the Use of Cloud Computing for Scientific Workflows*. IEEE Fourth International Conference on eScience (eScience 2008), Indianapolis, USA, 7-12 Dec. 2008.

[17] Job Submission Description Language (JSDL) Specification, Version 1.0, http://www.gridforum.org/documents/GFD.56.pdf

[18] D. Nurmi, R. Wolski, Ch. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov. *The Eucalyptus Open-source Cloud-computing System*. Proceedings of Cloud Computing and Its Applications 2008, Chicago, Illinois, October 2008.

[19] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar. *A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing.* in Proceedings of the 2005 European Grid Computing Conference (EGC 2005), Amsterdam, The Netherlands, February, 2005.

[20] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. B.-Y., W. Emmerich, F. Galan. *The RESERVOIR Model and Architecture for Open Federated Cloud Computing.*, IBM System Journal Special Edition on Internet Scale Data Centers, to appear.

[21] Amazon Simple Storage Services (S3), http://aws.amazon.com/s3/

[22] WS-Notification, http://www.ibm.com/developerworks/ webservices/library/specification/ws-notification

[23] Web Service Level Agreement (WSLA), http://www.research.ibm.com/WSLASpecV1-20030128.pdf