Technical University of Vienna
Information Systems Institute
Distributed Systems Group

Autonomic Resource Virtualization in Cloud-like Environments

Attila Kertesz (MTA SZTAKI)
Gabor Kecskemeti (MTA SZTAKI) and
Ivona Brandic
attila.kertesz@sztaki.hu
kecskemeti@sztaki.hu
ivona@infosys.tuwien.ac.at

TUV-1841-2009-04 December 14, 2009

Cloud computing is a newly emerged computing infrastructure that builds on the latest achievements of diverse research areas, such as Grid computing, Service-oriented computing, business process management and virtualization. An important characteristic of Cloud-based services is the provision of non-functional guarantees in the form of Service Level Agreements (SLAs), such as guarantees on execution time or price. However, due to system malfunctions, changing workload conditions, hard- and software failures, established SLAs can be violated. In order to avoid costly SLA violations, flexible and adaptive SLA attainment strategies are needed. In this paper we investigate the application of autonomic computing to SLA-based resource virtualization considering a three-layered Cloud based infrastructure including agreement negotiation, service brokering and deployment using virtualization. For each layer we exemplify how the principles of autonomic computing can be applied to achieve component self-management.

Keywords: Self-management, SLA negotiation, Service Brokering, On-demand deployment, Resource virtualization

Argentinierstr. 8/184-1
A-1040 Vienna, Austria
phone: +43 1 58801-18402
fax: +43 1 58801-18491
URL: http://www.infosys.tuwien.ac.at/

Autonomic Resource Virtualization in Cloud-like Environments

Attila Kertesz[1], Gabor Kecskemeti[1] and Ivona Brandic[2]

[1] MTA SZTAKI Computer and Automation Research Institute
H-1518 Budapest, P. O. Box 63, Hungary
{attila.kertesz,kecskemeti}@sztaki.hu
[2] TU Vienna
1040 Vienna, Argentinierstr. 8/181-1, Austria
ivona@infosys.tuwien.ac.at

Abstract. Cloud computing is a newly emerged computing infrastructure that builds on the latest achievements of diverse research areas, such as Grid computing, Service-oriented computing, business process management and virtualization. An important characteristic of Cloud-based services is the provision of non-functional guarantees in the form of Service Level Agreements (SLAs), such as guarantees on execution time or price. However, due to system malfunctions, changing workload conditions, hard- and software failures, established SLAs can be violated. In order to avoid costly SLA violations, flexible and adaptive SLA attainment strategies are needed. In this paper we investigate the application of autonomic computing to SLA-based resource virtualization considering a three-layered Cloud based infrastructure including agreement negotiation, service brokering and deployment using virtualization. For each layer we exemplify how the principles of autonomic computing can be applied to achieve component self-management.

Key words: Self-management, SLA negotiation, Service Brokering, On-demand deployment, Resource virtualization

1 Introduction

Grid Computing [6] has succeeded in establishing production Grids serving various user communities all around the world. Cloud Computing [2] is a novel infrastructure that focuses on commercial resource provision and virtualization. Both Grids and Service Based Applications (SBAs) already provide solutions for executing complex user tasks, but they are still lacking non-functional guarantees. The newly emerging demands of users and researchers call for expanding service models with business-oriented utilization (agreement handling) and support for human-provided and computation-intensive services [2]. Providing guarantees in the form of Service Level Agreements (SLAs) are highly studied in Grid Computing [10, 1, 3]. Nevertheless in Clouds, infrastructures are also represented as a service that are not only used but also installed, deployed or replicated with the help of virtualization. These services can appear in complex business processes, which further complicates the fulfillment of SLAs in Clouds. For example, due to changing components, workload and external conditions, hardware and software failures, already established SLAs may be violated. Frequent user interactions with the system during SLA negotiation and service executions (which are usually necessary in case of failures), might turn out to be an obstacle for the success of Cloud Computing. Thus, the development of the appropriate strategies for autonomic SLA attainment represents an emerging research issue. Autonomic Computing is one of the candidate technologies for the implementation of SLA attainment strategies. Autonomic systems require high-level guidance from humans and decide, which steps need to be done to keep the system stable [17]. Such systems constantly adapt themselves to changing environmental conditions. In our previous work [16] we presented a unified service architecture (called SLA-based Resource Virtualization – SRV) built on a three-layered Cloud-based infrastructure including agreement negotiation, brokering and service deployment/virtualization layer combined with business-oriented utilization used for SLA handling. In this paper we further examine this architecture and investigate how to apply principles of Autonomic Computing to the basic components of the SLA-based virtualization architecture in order to cope with changing user requirements and on demand failure handling.

The main contributions of this paper are: (i) the *application of principles of autonomic computing* to the SLA-based resource virtualization architecture, and (ii) *demonstration* of the application of autonomic computing based on selected *case studies* for meta-negotiation, meta-brokering, brokering and automatic service deployment. In the following section we summarize related works, in Section 3 we introduce the overall architecture and define the participating components. In Section 4 the components of the three problem areas are detailed, and in Section 5 we demonstrate the usage of the presented infrastructure in a car manufacturing scenario. Finally Section 6 concludes the paper.

2 Related work

Though cloud-based service execution is rarely studied, some related works have already started to investigate, how business needs and more dynamicity could be represented in service execution. Most of related works consider either virtualization approaches [5, 12, 7] without taking care of agreements or concentrates on SLA management neglecting the appropriate resource virtualizations [14, 3]. Works presented in [11, 9] discuss incorporation of SLA-based resource brokering into existing Grid systems, but they do not deal with virtualization. The Rudder framework [8] facilitates automatic Grid service composition based on semantic service discovery and space based computing.

Lee et al. discusses application of autonomic computing to the adaptive management of Grid workflows [20] with MAPE (Monitoring, Analysis, Planning, Execution) decision making [17], but they also neglect deployment and virtualization.

Regarding meta-brokering, LA Grid [13] developers aim at supporting grid applications with resources located and managed in different domains. They define broker instances, each of them collects resource information from its neighbors and save the information in its resource repository. The Koala grid scheduler [4] was redesigned to inter-connect different grid domains. They use a so-called delegated matchmaking (DMM), where Koala instances delegate resource information in a peer-2-peer manner. Gridway introduced a Scheduling Architectures Taxonomy [23], where Gridway instances can communicate and interact through grid gateways. These instances can access resources belonging to different Grid domains. Comparing the previous approaches, we can see that all of them use high level brokering that delegate resource information among different domains, broker instances or gateways. These solutions are almost exclusively used in Grids, they cannot co-operate with different brokers operating in pure service-based or cloud infrastructures. On the contrary, our proposed Meta-Broker can manage diverse, separated brokers.

Current deployment solutions do not leverage their benefits on higher level. For example the Workspace Service (WS) [5] as a Globus incubator project supports wide range of scenarios involving virtual workspaces, virtual clusters and service deployment from installing a large service stack to deploy a single WSRF service if the Virtual Machine (VM) image of the service is available. It is designed to support several virtual machines. The XenoServer open platform [12] is an open distributed architecture based on the XEN virtualization technique aiming at global public computing. The platform provides services for server lookup, registry, distributed storage and a widely available virtualization server. Also the VMPlants [7] project proposes an automated virtual machine configuration and creation service which is heavily dependent on software dependency graphs, but this project stays within cluster boundaries.

3 Application of the principles of autonomic computing to SLA-based resource virtualization (SRV) architecture

In this section we introduce the basic principles of autonomic computing, and describe the application of the autonomic computing to an SLA-based resource virtualization environment in Clouds.

3.1 Autonomic Systems

Autonomic systems require high-level guidance from humans and decide, which steps need to be done to keep the system stable [17]. 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.

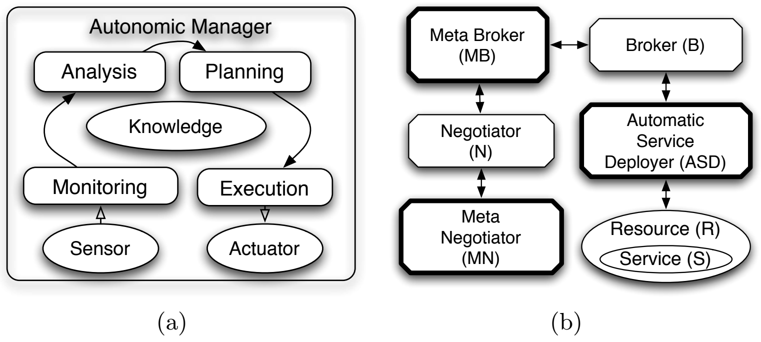

Fig. 1. (a) General architecture of an autonomic system and (b) the main components of SRV and their connections

An important characteristic of an autonomic system is an intelligent closed loop of control. As shown in Figure 1 the Autonomic Manager (AM) manages the element's state and behavior. It is able to sense state changes of the managed resources and to invoke appropriate set of actions to maintain some desired system state. Typically control loops are implemented as MAPE (monitoring, analysis, planning, and execution) functions [17]. 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*.

3.2 Autonomically managed SLA-based resource virtualization (SRV) approach

In our previous work [16] we presented a unified service architecture (SRV) that builds on three main areas: agreement negotiation, brokering and service deployment using virtualization. We suppose that service providers and service consumers meet on demand and usually do not know about the negotiation protocols, document languages or required infrastructure of the potential partners. The general architecture is highlighted in Figure 2. The relevant actors of this architecture are:

- MN – Meta-Negotiator: A component that manages Service-level agreements. It mediates between the user and the Meta-Broker, selects appropriate protocols for agreements; negotiates SLA creation, handles fulfillment and violation.
- MB – Meta-Broker: Its role is to select a broker that is capable of deploying a service with the specified user requirements, and propagate negotiation processes to brokers (by acting as a negotiator).
- B – Broker: It interacts with virtual or physical resources, and in case the required service needs to be deployed it interacts directly with the ASD.

- ASD – Automatic Service Deployment: It installs the required service on the selected resource on demand.
- S – Service: The service that users want to deploy and/or execute.
- R – Resource: Physical machines, on which virtual machines can be deployed/installed.

In our SRV approach users describe the requirements for an SLA negotiation on a high level using the concept of meta-negotiations. During the meta-negotiation only those services are selected, which understand specific SLA document language and negotiation strategy or provide a specific security infrastructure. After the meta-negotiation process, a meta-broker selects a broker that is capable of deploying a service with the specified user requirements. Thereafter, the selected broker negotiates with virtual or physical resources – with the help of ASD – using the requested SLA document language and the specified negotiation strategy. Once the SLA negotiation is concluded, service can be deployed on the selected resource using the virtualization approach.

In this paper we focus on illustrating how autonomic computing can be applied in the SRV components of the architecture. Figure 2 shows the autonomic management interfaces and connections of the components.

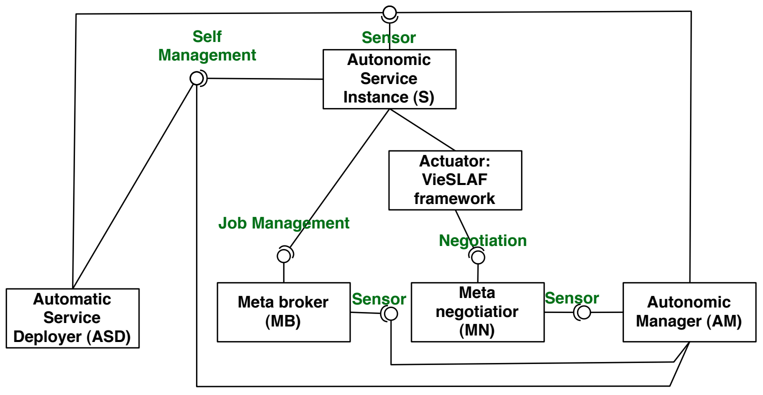

Fig. 2. Autonomic components in SRV.

We distinguish three types of interfaces: the *job management interface*, the *negotiation interface* and the *self-management interface*.

Negotiation interfaces are typically used by the monitoring processes of brokers and meta-brokers during the negotiation phases of the service deployment process. Self-management is needed 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 [16].

Job management interfaces are necessary for the manipulation of services during execution, for example for the upload of input data, or for the download of output data, and for starting or canceling job executions. Job management interfaces are provided by the service infrastructure and are automatically utilized during the service deployment and execution processes.

In the following we focus on the self-management interface. The *Autonomic manager* is notified about the system malfunction through appropriate sensors (see Figure 2). This interface specifies operations for sensing changes of the desired state and for reacting to that changes. Sensors can be activated using some notification approach (e.g. implemented by the WS-Notification standard). Sensors may subscribe for specific topic, e.g. violation of the execution time. Based on the measured values as demonstrated in [18] notifications are obtained, if execution time is violated or seems to be violated very soon. After the activation of the control loop, i.e. propagation of the sensed changes to the appropriate component, the service actuator reacts and invokes proper operations, e.g. migration of resources. An example software actuator used for the meta-negotiations is the VieSLAF framework [18], which bridges between the incompatible SLA templates by executing the predefined SLA mappings. Examples of the control loops are discusses in Figure 3. Based on various malfunction cases, the autonomic manager propagates the reactions to the *Meta negotiatiator*, *Meta-broker* or *Automatic Service Deployer*. We discuss the closed loop of control using the the example with meta-negotiations.

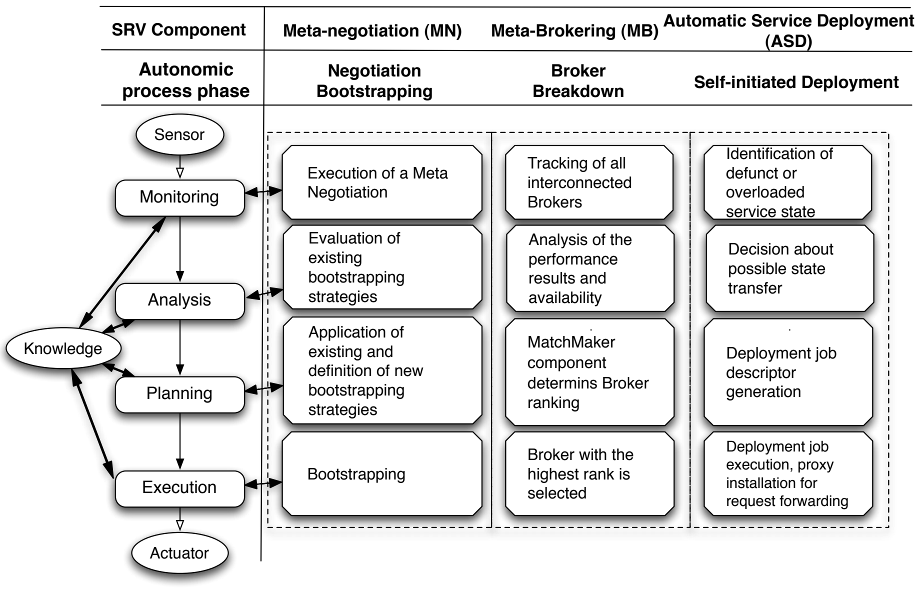
SRV Component
Meta-negotiation (MN)
Meta-Brokering (MB)
Automatic Service Deployment (ASD)
Autonomic process phase
Negotiation Bootstrapping
Broker Breakdown
Self-initiated Deployment
Sensor
Monitoring
Analysis
Planning
Execution
Actuator
Knowledge
Execution of a Meta Negotiation
Evaluation of existing bootstrapping strategies
Application of existing and definition of new bootstrapping strategies
Bootstrapping
Tracking of all interconnected Brokers
Analysis of the performance results and availability
MatchMaker component determins Broker ranking
Broker with the highest rank is selected
Identification of defunct or overloaded service state
Decision about possible state transfer
Deployment job descriptor generation
Deployment job execution, proxy installation for request forwarding

Fig. 3. Use cases for applying the autonomic processes to SRV

4 Autonomic components in SRV

In this section we investigate the SRV architecture considering self-management capabilities. We discuss autonomic capabilities using representative use cases for each SRV layer. Figure 3 summarizes the SRV components and the corresponding self-management examples. In section 4.1 we discuss the meta-negotiation component and present autonomic negotiation bootstrapping. In Section 4.2 we discuss the brokering functionalities of SRV and present a self-management solution for dealing with broker failures. Finally, in section 4.3 we discuss autonomic service deployment and virtualization for handling resource and service failures.

4.1 Meta-negotiation

In this subsection we demonstrate, how the principles of autonomic computing can be applied to the self-management of the meta-negotiation process by presenting a case study for negotiation bootstrapping and a case study for an autonomic system based on service mediation. A taxonomy of possible faults in this part of the architecture, and the autonomic reactions to these faults are summarized in Table 1.

Table 1. Taxonomy of faults and autonomic reactions for Meta Negotiation.

Fault	**Autonomic reaction**
non matching SLA templates	SLA Mapping [18]
non matching SLA languages	bootstrapping [19]
lack of adequate offers or providers	lookup for additional meta-negotiation repositories
inconsistencies in SLA templates	initiate new SLA mappings

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. 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 [2]. 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 3 depicts how the principles of autonomic computing can be applied to negotiation bootstrapping [19]. 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. (More on meta negotiation can be read in [1].) 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 (e.g. in order to bootstrap between WSLA and WS-Agreement).
- **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.

Service Mediation Case Study. The principles of autonomic computing can also be applied 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.

4.2 Service brokering

In this subsection we are focusing on the brokering components of the SRV architecture and demonstrate, how the principles of autonomic computing can be applied to these components. Brokers are the basic components that are responsible for finding the required services with the help of ASD. This task requires various activities, such as service discovery, matchmaking and interactions with information systems and service registries. In our architecture brokers need to interact with ASD and use adaptive mechanisms in order to fulfill agreements.

A higher-level component is also responsible for brokering in SRV: the Meta-Broker. *Meta-brokering* means a higher level resource management that utilizes existing resource or service brokers to access various resources. In a more generalized way, it acts as a mediator between users or higher level tools (e.g. meta negotiators or workflow managers) and environment-specific resource managers. The main tasks of this component are: to *gather* static and dynamic broker properties (availability, performance, provided and deployable services, resources, and dynamic QoS properties related to service execution), to *interact* with MN to create agreements for service calls, and to *schedule* these service calls to lower level brokers, i.e. match service descriptions to broker properties (which includes broker provided services). Finally the service call needs to be *forwarded* to the

Table 2. Taxonomy of faults and autonomic reactions in Service brokering.

Fault	Autonomic reaction
physical resource failure	new service selection
service failure	new service selection
wrong service response	new service selection
broker failure	new broker selection
no service found by some broker	initiate new service deployment
no service found by some broker	new broker selection
broker overloading	initiate new broker deployment
meta-broker overloading	initiate new meta-broker deployment

selected broker. More information on the components of the Meta-Broker can be read in [16]. Autonomic behaviour is needed by brokers basically in two cases: the first one is to survive failures of lower level components, the second is to regain healthy state after local failures. A taxonomy of the sensable failures and the autonomic reactions of the appropriate brokering components are gathered and shown in Table 2. The fourth case, the broker failure, is detailed in the next paragraph. To overcome some of these difficulties brokers in SRV use the help of the ASD component to re-deploy some services. Its autonomic operation is detailed in the next subsection.

Broker Failures Case Study. In the following we present a case study showing how autonomic principles can be applied to the Meta-Broker to handle broker failures:

– **Monitoring.** During the monitoring phase all the interconnected brokers are tracked: the IS Agent component of the Meta-Broker gathers state information about the brokers. The Matchmaker component also incorporates a feedback-based solution to keep track of the performances of the brokers.
– **Analysis.** During the analysis phase the Information Collector of the Meta-Broker is queried for broker availability and performance results.
– **Planning.** In case of incoming service request the MatchMaker component determins the ranking of the broker according to their performance data gathered in the previous phases. In case of a broker failure the ranks are recalculated and the failed broker is skipped.
– **Execution.** Finally, during the execution phase the broker with the highest rank is selected for invocation.

4.3 Service deployment and virtualization

Automatic Service Deployment (ASD – [15]) is a *higher-level* service management concept, which provides the dynamics to SBAs, e.g. during the SBA's lifecycle services can appear and disappear without the disruption of their overall behavior. The master copies of all *deployable services* should be stored in the repository. In this context the master copy means everything what is needed in order to deploy a service on a selected site – which we call a *virtual appliance*

(VA). A VA could be either defined by an external entity or the ASD should be capable of acquiring it from an already running system. The repository entries help determine which services are available for deployment and which are the static ones.

In SRV there is a bidirectional connection between the ASD and the service brokers. First the service brokers could instruct ASD to *deploy* a new service. This scenario was discussed in detail in [16]. However, deployments could also occur independently from the brokers as explained in the following. After these deployments the ASD has to *notify* the corresponding service brokers about the infrastructure changes. This notification is required, because information systems cache the state of the SBA for scalability. Thus even though a service has just been deployed on a new site, the broker will not direct service requests to the new site. This is especially needed when the deployment was initiated to avoid an SLA violation.

Table 3. Taxonomy of faults and autonomic reactions in Self-Initiated Deployment.

Fault	**Autonomic reaction**
Degraded service health state	Service reconfiguration
Reconfiguration fails	Initiate service cloning with state transfer
Defunct service	Initiate service cloning
Service decommissioned	Offer proxy
Proxy lifetime expired	Decommission service proxy

Self-initiated Deployment. Here we provide some examples how third party entities that can initiate deployment. Firstly, a BPEL engine could initiate deployment of a service that is going to be used by the currently executed process or workflow. Secondly, a prediction engine can initiate preemptive deployments to avoid future peak service usages. Thirdly, it is possible to implement a service container that could deploy services in order to serve requests to previously unknown services. Finally, services with self-management interfaces (seen on figure 2) could identify erroneous situations that could be solved by deploying an identical service on another site. A summary of these possible faults and the autonomic reactions can be seen in Table 3. We call this autonomous technique the self-initiated deployment, and in the following we describe its autonomous behavior:

– **Monitoring.** During the monitoring phase the ASD identifies two critical situations. Firstly, when the autonomous service instance gets defunct (it is not possible to modify the service instance so that it could serve requests again). And secondly, the service instance could also get overloaded on such an extent that the underlying virtual machine cannot handle more requests.
– **Analysis.** The ASD first decides whether the service initiated deployment is required (because it was overloaded) or the its replication is necessary (because it has became defunct). First in both cases the ASD identifies the

service's virtual appliance to be deployed, then in the latter case the ASD also prepares for state transfer of the service before it is actually decommissioned.

– **Planning.** After the ASD identified the service it generates a deployment job and requests its execution from the service broker.
– **Execution.** If a service is decommissioned on the original site, then a service proxy is placed instead on the site. This proxy forwards the remaining service requests to the newly deployed service using WS-Addressing. The proxy decommissions itself when the frequency of the service requests to the proxy decreases under a predefined value.

5 Scenario for autonomic arrangements of car assembly testing services with the SRV architecture

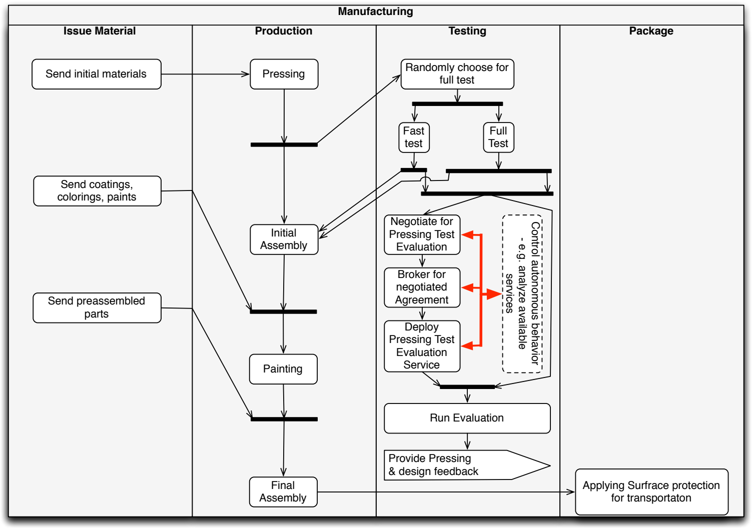
Manufacturing
Issue Material
Production
Testing
Package
Send initial materials
Pressing
Randomly choose for full test
Fast test
Full Test
Send coatings, colorings, paints
Initial Assembly
Negotiate for Pressing Test Evaluation
Broker for negotiated Agreement
Deploy Pressing Test Evaluation Service
Control autonomous behavior - e.g. analyze available services
Send preassembled parts
Painting
Run Evaluation
Provide Pressing & design feedback
Final Assembly
Applying Surfrace protection for transportaton

Fig. 4. Example scenario with the SRV architecture.

In this section we demonstrate the utilization of our proposed SRV architecture. We use the Auto Inc. scenario developed by the S-Cube project [21]. We extend this scenario, and focus on its manufacturing step (see the step called M1.3 on figure 4.9 in [22]). This step can be detailed on arbitrary levels, Figure 4 however does not plan to give a full overview on the car manufacturing process. The main objective of this figure is to show the relation between the production and testing processes of the manufacturing step. The production process is split to several phases by Auto Inc. After the completion of every phase of the production process, there is an option for testing the partially ready product. On Figure 4 we choose the decision after the phase *Pressing* to show the initiation of the testing process. This decision however happens after every phase (e.g. initial assembly or painting), and the corresponding testing process could be also initiated after their completion.

In the following paragraphs we are going to discuss what happens after the decision is made that the scenario takes the alternative course towards testing. The available computing resources are limited by the Auto Incs infrastructure, so the services evaluating the test results are autonomously managed by our proposed three layered service infrastructure (SRV). The infrastructure layers are negotiation, brokering, and deployment. These layers adapt the Auto Incs infrastructure to fit the current needs. In case the adaptation is not possible within the boundaries of Auto Inc. the infrastructure layers could introduce external Cloud computing [2] resources for the actual demands.

When a new testing evaluation request is formulated the testing process, it initiates the meta-negotiation to propagate/translate the request details towards lower levels. The meta-negotiation service applies self-management during the negotiation bootrstrapping procedure as follows. Before using a service, the service consumer and the service provider have to establish an electronic contract defining the terms of use. 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. Automatic bootstrapping between different protocols and contract formats increases the number of services a consumer may negotiate with. Consequently, the full potential of publicly available services could be exploited. The management is done through following steps: as a prerequisite of the negotiation bootstrapping Auto Incs testing process specifies a meta-negotiation document describing the requirements of a negotiation, as for example required negotiation protocols, required security infrastructure, provided document specification languages, etc. The autonomic meta-negotiation management steps are described in the following:

1. All candidate services are selected, where negotiation is possible or bootstrapping is required.
2. The knowledge base is queried and potential bootstrapping strategies are found (e.g. in order to bootstrap between WSLA and WS-Agreement). In case of missing bootstrapping strategies users can define new strategies in a semi-automatic way.
3. Finally, the negotiation is started by utilizing appropriate bootstrapping strategies.

After the negotiation has completed the brokering services further process the test evaluation request. Brokers are the basic services that are responsible for finding the requested services with the help of a deployer service. This task requires various activities, such as service discovery, matchmaking and interactions with information systems and service registries. Autonomic behaviour is needed by brokers basically in two cases: the first one is to survive failures of lower level components (e.g. broker, job execution), the second is to regain healthy state after local failures (e.g. overloading of available testing services, breakdown of the used services). To overcome these difficulties, the meta-broker uses the self-management techniques presented in 4.2, and the brokers may rely on the deployer component, the automatic Service Deployment (ASD). In the testing phase of the Auto Inc. scenario, deployment or decomission of a service instance on local resources or on the Cloud could also occur independently from the brokers. Therefore after these deployments the ASD notifies the corresponding service brokers about the infrastructure changes. This notification is required, because the information system caches the states for scalability. Thus even though a service has just been deployed on a new site, the broker will not direct service requests to the new site. This is especially needed when the deployment was initiated to avoid an SLA violation.

Services with self-management interfaces could identify erroneous situations that could be solved by deploying an identical service on another site (e.g. a testing service replication to a Cloud resource due to increased service requests). We call this autonomous technique the self-initiated deployment, and in the following we describe its autonomous behavior:

1. The ASD monitors the occurance of critical situations. First of all it looks for service instances that became defunct because they cannot modify themselves so that they can serve future requests properly. Secondly, a service instance could also get overloaded on such an extent that the underlying resources cannot handle more requests.
2. In critical situations the ASD first decides whether the service initiated deployment is required (because the service was overloaded) or replication is necessary (because the service became defunct). First in both cases the ASD identifies the service's virtual appliance (or master copy) to be deployed, then in the latter case the ASD also prepares for state transfer of the service before it is actually decommissioned.
3. The ASD generates a deployment job for the service broker layer. This job refers to the sevice to be deployed and the state the deployment service needs to resume. As a result the test evaluation service can serve the testing processs request.
4. Optional behavior in case a service is decommissioned: A service proxy is placed on the computing resource instead of the decommissioned service instance. This proxy forwards the remaining service requests to the newly deployed service. The proxy decommissions itself when the frequency of the service requests to the proxy decreases under a predefined value.

Finally after the requested test evaluation service is identified (or even deployed) the test process can invoke the evaluation service and retrieve the results to provide feedback for the production and design processes.

6 Conclusions

In this paper we described the first steps towards an autonomic architecture for SLA-based resource virtualization with on-demand service deployment. The proposed solution incorporates enhancements of a meta-negotiation component for generic SLA management, a meta-brokering component for diverse broker management and an automatic service deployment for resource virtualization on the Cloud. We have discussed how the principles of autonomic computing can be incorporated to the SRV architecture, and demonstrated MAPE actions with the corresponding case studies. Our future work aims at defining a taxonomy of events for sensors and actuators in the Automatic Manager of SRV.

7 Acknowledgement

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube), and by the Vienna Science and Technology Fund (WWTF) under agreement ICT08-018, FoSII – Foundations of Self-governing ICT Infrastructures.

References

1. I. Brandic, D. Music, S. Dustdar, S. Venugopal, and R. Buyya. Advanced qos methods for grid workflows based on meta-negotiations and sla-mappings. In *The 3rd Workshop on Workflows in Support of Large-Scale Science*, pages 1–10, November 2008.
2. 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 Generation Computer Systems*, 2009.
3. M. Q. Dang and J. Altmann. Resource allocation algorithm for light communication grid-based workflows within an sla context. *Int. J. Parallel Emerg. Distrib. Syst.*, 24(1):31–48, 2009.
4. A. Iosup, T. Tannenbaum, M. Farrellee, D. Epema, and M. Livny. Inter-operating grids through delegated matchmaking. *Sci. Program.*, 16(2-3):233–253, 2008.
5. K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the grid. *Sci. Program.*, 13(4):265–275, 2005.
6. C. Kesselman and I. Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.
7. I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2004. IEEE Computer Society.
8. Z. Li and M. Parashar. An infrastructure for dynamic composition of grid services. In *GRID '06: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pages 315–316, Washington, DC, USA, 2006. IEEE Computer Society.

9. 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)*, February 2005.
10. M. Parkin, D. Kuo, J. Brooke, and A. MacCulloch. Challenges in eu grid contracts. In *Proceedings of the 4th eChallenges Conference*, pages 67–75, 2006.
11. D. M. Quan and J. Altmann. Mapping a group of jobs in the error recovery of the grid-based workflow within sla context. *Advanced Information Networking and Applications, International Conference on*, 0:986–993, 2007.
12. D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: Accountable execution of untrusted programs. In *In Workshop on Hot Topics in Operating Systems*, pages 136–141, 1999.
13. I. Rodero, F. Guim, J. Corbalan, L. Fong, Y. Liu, and S. Sadjadi. Looking for an evolution of grid scheduling: Meta-brokering. In *Grid Middleware and Services Challenges and Solutions*, pages 105–119. Springer US, 2008.
14. M. Surridge, S. Taylor, D. De Roure, and E. Zaluska. Experiences with gria – industrial applications on a web services grid. In *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*, pages 98–105, Washington, DC, USA, 2005. IEEE Computer Society.
15. G. Kecskemeti, P. Kacsuk, G. Terstyanszky, T. Kiss, T. Delaitre. Automatic service deployment using virtualisation. In *Proceedings of 16th Euromicro International Conference on Parallel, Distributed and network-based Processing.* IEEE Computer Society. February 2008.
16. A. Kertesz, G. Kecskemeti and I. Brandic. An SLA-based Resource Virtualization Approach For On-demand Service Provision. In *Proceedings of 3rd International Workshop on Virtualization Technologies in Distributed Computing.* ACM. June, 2009.
17. J.O. Kephart, D.M. Chess. The vision of autonomic computing. *Computer* . 36:(1) pp. 41-50, Jan 2003.
18. I. Brandic, D. Music, P. Leitner, S. Dustdar. VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management. In *the 6th International Workshop on Grid Economics and Business Models 2009 (Gecon09)*, 2009.
19. I. Brandic, D. Music, S. Dustdar. Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services. In *Proceedings of Grids meet Autonomic Computing Workshop.* ACM. June, 2009.
20. K. Lee, N.W. Paton, R. Sakellariou, E. Deelman, A.A.A. Fernandes, G. Mehta. Adaptive Workflow Processing and Execution in Pegasus. In *Proceedings of 3rd International Workshop on Workflow Management and Applications in Grid Environments*, pages 99–106, 2008.
21. S-Cube project website. http://www.s-cube-network.eu/, 2008.
22. The S-Cube deliverable CD-IA-2.2.2. Collection of Industrial Best Practices, Scenarios and Business Cases. Project deliverable, 2008. http://www.s-cube-network.eu/results/deliverables/wp-ia-2.2/CD-IA-2.2.2_Collection_of_industrial_best_practices_scenarios_and_business_cases.pdf
23. T. Vazquez, E. Huedo, R. S. Montero, and I. M. Llorente. Evaluation of a utility computing model based on the federation of grid infrastructures. In *Euro-Par 2007 Parallel Processing*, pages 372–381. Springer Berlin / Heidelberg, 2007.