SLA MANAGEMENT FOR COMPOSITE INFRASTRUCTURE AS A SERVICE

CONTENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2010-2012) under grant agreement n° 248657

A NEW ERA FOR INFRASTRUCTURE AS A SERVICE (IAAS)

Applications and infrastructure services are no longer single, isolated, remote servers, virtual machines or links. They are composed of resources from various providers, such that interoperability across cloud providers is a challenge[1]. We refer to this new era of IaaS as *composite IaaS* – the involvement of multiple infrastructure resource types and management domains in delivering a single infrastructure service. Furthermore, application topologies continue to become more complex and heterogeneous as service-oriented design becomes de-facto[2]. For these reasons the interdependencies between application deployment, usage and network capabilities need to be considered during service planning and provisioning, leading to challenges for Service Level Agreement (SLA) management. We have identified the need for an approach to SLA Management that supports autonomous collaboration and cross stratum optimization (CSO)[3], where a stratum refers to a layer or domain, as shown in

Figure 1. These two objectives are not straightforward to achieve together, as autonomy is the ability of an entity to control access to its information and resources, while CSO requires access to information across layers.

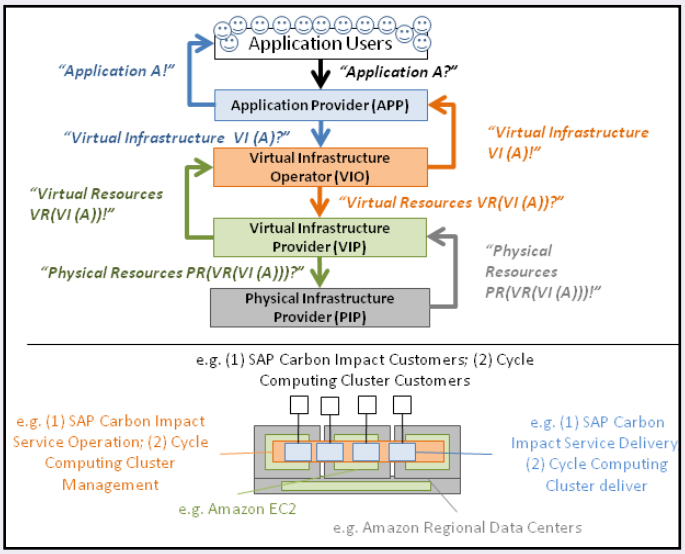

Figure 1. An illustration and example of the cross-layer and cross-domain problem for composite IaaS

"Applications and infrastructure services are no longer single, isolated, remote servers, virtual machines or links."

As an example of how the landscape of application topologies and infrastructure services has developed, consider the SAP Carbon Impact[4] application delivered as an on-demand application service for customers. The infrastructure behind Carbon Impact is not a concern for the application users. However, the SAP Carbon

[1]David Bernstein, Erik Ludvigson, Krishna Sankar, Steve Diamond, Monique Morrow, "Blueprint for the Intercloud - Protocols and Formats for Cloud Computing Interoperability," iciw, pp.328-336, 2009 Fourth International Conference on Internet and Web Applications and Services, 2009

[2] Longji Tang; Jing Dong; Yajing Zhao; Liang-Jie Zhang; , "Enterprise Cloud Service Architecture," Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pp.27-34, 5-10 July 2010

[3] Javier Jiménez Chico,Telefonica, representing the GEYSERS project at the Cloud Computing and Cross Stratum Optimization Workshop, June 2011. Available: http://www.cccso.net/CSOWorkshopFinal.html

[4] SAP's Carbon Impact On-Demand software enables your business to manage carbon, energy in facilities, and product lifecycle assessments. Online: http://www.sapenvironmentalimpact.com/

Impact service delivery team operates and maintains a virtual infrastructure composed of Amazon EC2[5] machine instances, which are virtual resources hosted in specific locations across Amazon's regional data centers. As a second example Cycle Computing[6] provides a cluster service primarily for scientific experiments and analysis, such as molecular modelling. They have deployed a cluster called Nekomata[7] of over 30,000 cores at a cost of 1,3K per hour. This includes 30 TB of RAM and 3,809 XLarge EC2 instances, proving that IaaS is for various sizes of applications and is growing in acceptance.

Amazon currently has the largest share of the Infrastructure as a Service (IaaS) market[8], given their technology leadership, reputation and operational regions. However, consider if Amazon's data centers experiences a failure and downtime, as occurred in April 2011, when many major websites and services became unavailable[9]: there is potential loss for long-running scientific work done by Cycle Computing's pharmaceutical customers and for SAP's Carbon Impact customers. Not only is there a need for technical recovery to resume productivity, but there is a need to recover financially with regard to shared liability amongst service providers. This is where SLAs become more than just legal or sales agreements. They inform the process of resource provisioning, error preemption and recovery.

THE MULTI-DIMENSIONED SLA MANAGEMENT PROBLEM FOR COMPOSITE IAAS

The initiative to develop a novel IaaS service delivery model makes the SLA Management problem more of a concern. Figure 2 depicts this concern as a multi-dimensional problem, where various layers, objectives, resource types and autonomous entities are involved.

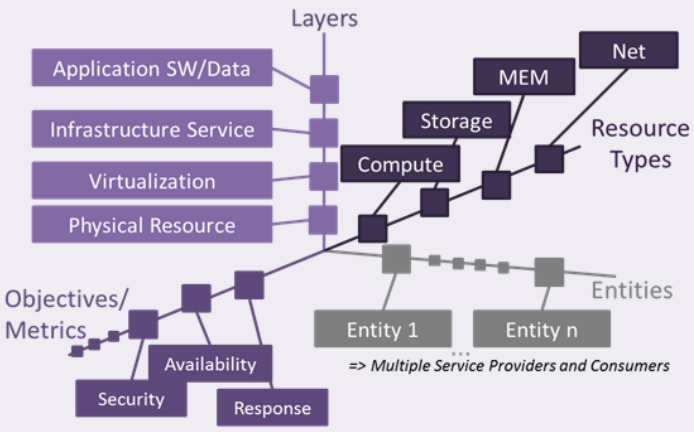

Figure 2. The multiple dimensions of the SLA management problem for composite IaaS.

"Each layer has its own operational requirements, capabilities, constraints and potentially different administrators that can act autonomously…"

[5] Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. Online: http://aws.amazon.com/ec2/

[6] Cycle Computing provides software tools & solutions to manage HPC workflows on in-house or cloud clusters. Online: http://www.cyclecomputing.com/

[7] New CycleCloud HPC Cluster Is a Triple Threat: 30000 cores, $1279/Hour, & Grill monitoring GUI for Chef. Blog 19 September 2011. http://blog.cyclecomputing.com/2011/09/new-cyclecloud-cluster-is-a-triple-threat-30000-cores-massive-spot-instances-grill-chef-monitoring-g.html

[8] A listing of top 10 cloud computing companies in 2011 - Amazon came out on top: http://www.cloudcomputing-companies.org/

[9] Why Amazon's cloud Titanic went down. By David Goldman, staff writer. April 22, 2011: 5:37 PM ET. Online: http://money.cnn.com/2011/04/22/technology/amazon_ec2_cloud_outage/index.htm

The first dimension of the problem is the multiple system layers that are involved in delivering a combined IT and Net VI as a service. Each layer has its own operational requirements, capabilities, constraints and potentially different administrators that can act autonomously, although upper layers are dependent on lower layers for their runtime. For example a Virtual Machine is only as powerful as the Physical Machine that hosts it. Subsequently, an Infrastructure Service is only as responsive and available as the virtual resources and virtualization management behind its specification. Finally, applications deployed using infrastructure services are limited by the capabilities of the service with respect to operation but also with respect to manageability: what can be monitored and controlled.

This leads to the second dimension: different resource types. The objectives in an SLA indirectly require different classes of resources to be provisioned and configured. The resources in IaaS are compute (which refers to CPU capabilities, types and number of cores), RAM, Storage and Networking. The performance and availability of application and infrastructure services has dependencies on all these resources types. For example, if an application states an SLO of 2s response time per transaction, this round-trip-time includes the Input/Output (I/O) of the CPU (as well as the cache attached), RAM, Disks and network throughput. The virtualization layer adds additional overhead to I/O depending on the type of virtualization used[10].

A third dimension of the problem is need to trade off between multiple objectives and QoS metrics. For example, consider the case of Amazon EC2 again; services are often replicated across multiple locations and there is the possibility to select specific regions based on the class of physical resource capabilities provided (for example, Amazon's Cluster Quadruple GPU XLarge comes with 2 Nvidia Tesla "Fermi" M2050 GPUs per node). The choice to replicate and specialise comes with a price but is seen as necessary to achieve a certain level of availability, reliability and performance, especially required in an Enterprise setting[11]. However, the choice to use IaaS has an inherent impact on security, as the physical, direct control of resources for applications shifts from the application provider to the infrastructure provider.

This then leads to the fourth dimension, where each entity has their own, autonomous operational objectives and service level objectives to be satisfied. That is, providers and consumers act freely within the context of resources and services they own and acquisition. Providers are free to (re)start, shutdown, configure and adjust resources and associated service instances as they see fit. Consumers are free to request or quit service access as they see fit, given the financial obligations of the service usage. This also means that providers are free to compare their penalties for not honouring SLAs against their own operational objectives such as cost minimisation[12]. We summarise this using 4 axioms for SLA management of composite IaaS:

1. **Monitoring is NOT transitive:** if A can monitor B and B can monitor C, this does not imply that A can monitor C. For example, an application provider can monitor metrics related to their application such as response time and usage but cannot monitor the state of the physical infrastructure hosting the virtual machines that execute the application.
2. **Knowledge is NOT transitive:** if A knows B (i.e. maintains logs of B) and B knows C, this does not imply that A knows C. For example, a customer of an application service does not necessarily know the virtual infrastructure being used or, moreover, where the application service is physically running.
3. **Control is NOT transitive:** if A controls B (i.e. can request or directly change the state of B) and A controls C, this does not imply that A can control C. For example the application provider can request

[10] Yiduo Mei, Ling Liu, Xing Pu, Sankaran Sivathanu, Xiaoshe Dong, "Performance Analysis of Network I/O Workloads in Virtualized Data Centers," IEEE Transactions on Services Computing, 14 June 2011.

[11] Ellahi, T., Hudzia, B., Li, H., Lindner, M. A. and Robinson, P. (2011) The Enterprise Cloud Computing Paradigm, in Cloud Computing: Principles and Paradigms (eds R. Buyya, J. Broberg and A. Goscinski), John Wiley & Sons

[12] Hui Li, Giuliano Casale, and Tariq Ellahi. 2010. SLA-driven planning and optimization of enterprise applications. In Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering (WOSP/SIPEW '10).

more CPU or RAM but the application user cannot. They can require faster response time but it is up to the application provider to address this appropriately.

4. **Affect IS transitive:** If A affects B and B affects C then A affects C. For example, consider the Amazon EC2 failure affected its AWS users as well as the users of applications running on EC2: the customers of their customers.

The axioms define what we refer to as the *hidden layer problem*, where the owners and providers of resources or services do not necessarily know the details of how and by whom their resources will be used. These 4 axioms make a centralised SLA Management solution ineffective. With these issues in mind we identify the following 3 requirements for composite IaaS SLA Management:

1. Maintain **autonomy** of providers and management domains, such that they can implement their own policies and operational objectives for their resources. They can maintain control over access, information release and manipulation of their managed resources.

2. **Convergence** of service level management such that the dependencies between resource types (i.e. service access, network, compute, storage) are well defined and management can be done as a single function and hence possibly implemented in a single solution. Reduced switching between management systems and consoles.

3. **Coordination** between providers and management domains for cooperatively handling events and alerts.

There are already many industrial solutions for SLA Management, which we discuss here towards deriving the essential elements for a reference architecture. *IBM Tivoli Service Level Advisor*[13] provides a way to define and record service level agreements based on the definitions of service level objective while it analyzes trends and tendencies so that potential problems can be identified and corrected before they occur. It supports notification of SLA violations and trends toward potential violations and generates and presents SLA reports. *HP Service Quality Manager*[14] offers real-time service-level and SLA monitoring, SLA reporting and is based on a technology-neutral service model. Data collection monitoring is performed using Service Adapters and it also performs detection of SLA breaches with action triggering. *Uptime software*[15] offers a complete SLA Management, being a monitoring and reporting solution supported by a comprehensive monitoring framework. It supports multiple SLA related functionalities such as proactive notification of SLA degradations, monitoring of physical, virtual or cloud applications, allows capacity planning and server consolidation, supports co-location, multi-datacenter, and remote monitoring, supports NetFlow network monitoring. Although Uptime's solution is targeted at a multi-domain environment, there are still some missing concepts for enabling effective SLA management of composite IaaS.

SLA COMPOSITION AND SPECIFICATION

In GEYSERS the concept of composite IaaS is realized by the definition of a Virtual Infrastructure (VI). To understand this concept, the Virtual Infrastructure Description Language (VXDL™)[16] exists as an information

[13] IBM® Tivoli® Service Level Advisor is designed to provide predictive service level management capabilities. Online: http://www-01.ibm.com/software/tivoli/products/service-level-advisor/

[14] HP OpenView Service Quality Manager automates the definition, configuration, the real-time monitoring and historical reporting. Online: http://h20229.www2.hp.com/products/sqm/index.html

[15] up:time's SLA engine is based on Gartner's SLA best practices, as well as customer ease-of-use feedback. Online: http://www.uptimesoftware.com/sla-management.php

[16] Virtual Infrastructure Description Language (VXDL™) is an open XML-based language that enables the modeling and description of virtual resources and virtual infrastructure. Online: http://www.lyatiss.com/technology/vxdl-language/

model and XML-based language. VXDL allows the formal description and configuration of the Virtual Infrastructure as a single object, the virtual resources and groups of resources composing it, the virtual network topology interconnecting individual virtual entities and the temporal attributes of all these elements. The VXDL language enables users and applications to submit requests of virtual infrastructures to Infrastructure Service Providers (InPs) using a well-defined format. A VXDL request represents an abstracted VI. When the VI is provisioned, a corresponding VXDL file which details the characteristics of the embedded VI can be generated. The figure below represents the VXDL request submission scenario. The VXDL files can be generated directly by users or by machines.

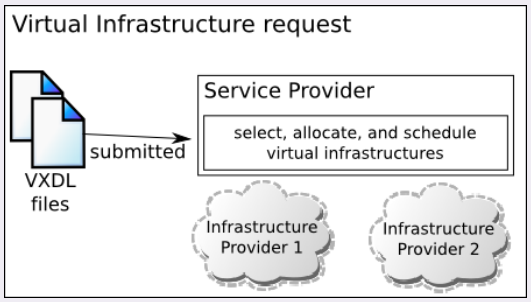

Figure 3. Virtual Infrastructure Description Language (VXDL)

"The VXDL language enables users and applications to submit requests of virtual infrastructures to Infrastructure Service Providers using a well-defined format."

This description of a VI as a realisation of composite IaaS does not change the fundamental definition of an SLA but it does change the roles and how they are involved in the SLA lifecycle[17]. An SLA occurs when a service consumer accepts specific service parameters on entering a contract with a service operator or provider. In composite IaaS the role of provider and consumer changes up and down the service delivery stack. An entity in the role consumer specifies *Service Level Objectives (SLOs)*, before or after the availability and capability of services and service providers are known. Providers declare their service capabilities and quality guarantees in the form of an advertisement, known as a *Service Level Agreement Template (SLAT).* Such templates act as the baseline for contractual agreement with customers, potentially in different classes. SLOs, SLATs and SLAs have basic requirements for structure and content and can hence be represented using the same information model (we then use the general term "SLA"):

1. **Parties** include individuals, organizations and roles involved in the agreement. The roles are typically consumer/customer and provider/operator, but can also include a third-party broker, an intermediate actor in the SLA management process. In GEYSERS these are PIP, VIP, VIO and APP, but can be extended to include other specialist roles.

2. **Functional description** of the service's purpose and capabilities. In the case of a technical, IT service, the functional description refers to the set of operations, methods and parameters. For example, the Web-Services Description Language (WSDL) provides a standard specification for SOAP-based web services. In the case of a network or connectivity service the functional description refers to path selection and bandwidth provisioning. GEYSERS does not use WSDL as we adopt a RESTFul approach[18], but we make note of the attributes of WSDL that are used in SLA lifecycles.

[17] Escalona, E., et al: GEYSERS: A Novel Architecture for Virtualization and Co-Provisioning of Dynamic Optical Networks and IT Services. In: ICT Future Network and Mobile Summit 2011, Santander, Spain (June 2011)

[18] Roland Kübert, Gregory Katsaros, and Tinghe Wang. 2011. A RESTful implementation of the WS-agreement specification. In Proceedings of the Second International Workshop on RESTful Design (WS-REST '11)

3. **Costs** to the consumer for receiving the service. The units for costs are defined by relating financial costs to utility functions of the resources consumed by the service. For example costs can be defined per requests, per volume of storage used, per user or on a fixed-term or unlimited basis.

4. **Guarantee or Quality of Service (QoS)** terms are the non-functional properties of the service. These properties include availability, performance, response time, reliability and security.

5. **Recovery** terms define what types of consumer-visible incidents the service can recover from using an event-action mapping. An event and action can also define compensation, stating what the consumer can rightfully demand from the service provider in return, should the functional or guarantee terms not be fulfilled.

The distinction between a SLA document and a service configuration request for a service infrastructure is fuzzy when dealing with on-demand service provision. The contents of an SLA are inevitably used as concrete configuration directives[19]. These parameterise the provisioning of resources, deployment of software and tuning of settings to enable effective operation of the service. A service's operation is *effective* if an acceptable trade-off is found between satisfying the functional and non-functional terms in the SLA and minimised expenses and operational costs for the consumer and provider. Providers need to keep their costs down so that they can offer an attractive service deal to consumers without sacrificing their business profitability.

GEYSERS SLA: This is a template for a GEYSERS Service Level Agreement

1. **<Provider>** *Name of provider and service being delivered (e.g. Amazon EC2)*
    1. **<Provider-Type>** *role or type of service provider (e.g. VIP)*
    2. **<Provider-Details>** *<Provides details including name, contact information and other properties of the entity that might be necessary for legal purposes – these details are included for completeness but are not assumed to have direct consequence on the handling of SLAs in GEYSERS>*
2. **<Consumer>** *Name of the consumer (e.g. SAP); recall SLAs are bilateral*
    1. **<Provider Type>** *Role of the consumer (e.g. VIO)*
    2. **<Provider Details>** *Contact details of the consumer (similar to provider)*
3. **<Level>** *Descriptor for distinguishing different levels of quality associated with a SLA or Template*
    1. **<Grade>** *The grade used to categorise the SLA or Template. Examples include Gold, Silver, Bronze or Priority, Intermediate, Basic. This can however be defined for the environment in question.*
        1. **<Cost>** *A cost associated with the grade of service*
            1. **<Value-per-Unit>** *The numeric value of the cost*
            2. **<Unit>** *The unit of cost e.g. Currency per month, Currency per transaction, Currency per GB*
        2. **<Description>** *A textual description of the grade categorisation for the SLA or Template*
    2. **<Access Point>** *A technical access point or URL to a provider of the service at the given quality level*
        1. **<Protocol>** *The communications protocol used for interacting with the provider e.g. HTTP, SMTP, PPP*
        2. **<Address>** *A unique reference to a logical or physical service access point for the*

[19] Philip Robinson, Alexandru-Florian Antonescu, et al. "Towards Cross Stratum SLA Management with the GEYSERS Architecture" To Appear - International Workshop on Cross-Stratum Optimization for Cloud Computing and Distributed Networked Applications, July 2012

provider

3. **<Function>** *The capabilities provided by a service at a given quality level*
    1. **<Type>** The nature of the function e.g. connectivity creation, storage volume creation
    2. **<Purpose>** *The need for the function e.g. provisioning*
    3. **<Operation>** *The operations that can be involved to execute the function e.g. create, read, update, delete*
        1. **<Operation-Type>** The data or item returned by the operation
        2. **<Parameters>** *The set of input data for the operation*
    4. **<Guarantee>** *The set of guarantees at a given service level e.g. performance, reliability, security*
        1. **<Metric-per-Guarantee>** *The metric used to measure the guarantee e.g. response time, mean time to failure, mean time to recovery, cryptographic suite*
        2. **<Upper Bound>** *The upper value for a guarantee*
        3. **<Certainty>** *The degree of certainty the provider promises*
        4. **<Lower Bound>** *The lower value for a guarantee*
    5. **<Recovery>** *A specification of actions given specific classes of incidents*
        1. **<Action>** *The type of action taken if an incident or class of event occurs e.g. restart, new, ignore*
        2. **<Event>** *Classes of events that represent incidents e.g. timeout, illegal access*

It is impractical to place a boundary on the number and type of SLA metrics that can be defined for a generic infrastructure service delivery platform, as such metrics are typically domain, application and service specific. The Open Data Center Alliance's Usage Models (ODAC-UM)[20], and in particular the Standard Units of Measurement for IaaS document treat the problem of putting together a set of metrics, both quantitative and qualitative, for supporting an objective and transparent comparison between different cloud infrastructure providers. This is similar to the objective of GEYSERS' classification of SLA metrics for the domain of virtual resource and infrastructure service providers. ODAC defined requirements related to Secure Federation, Automation, Common Management & Policy, and Transparency in cloud infrastructure providers. The ODAC Usage Model is designed to provide the attributes used to describe and measure the capacity, performance and quality of a cloud service, with respect to its compute, network and storage components, which are comparable to the metrics defined in GEYSERS.

SLA MANAGEMENT STRATEGIES FOR COMPOSITE IAAS

A service level management strategy is a specification of an approach or process for creating "the best" SLA given a set of SLOs and resource and service configuration possibilities. The implementation of strategies is the responsibility of the service provider. The provider's aim is to satisfy the SLOs of consumers without disrupting their internal operation goals – minimisation of operational costs, power consumption, burn-out of equipment and legal issues. There are three strategies identified: (1) the *bottom-up strategy* that is initiated by providers, (2) the *top-down strategy* that is initiated by consumers and (3) the mixed/negotiated strategy, which is a

[20] Open Data Center Alliance Usage Models - 8 models were published in June 2011. Online: http://www.opendatacenteralliance.org/ourwork/usagemodels

combination of 1 and 2, characterised by a series of message exchanges in order to reach a mutually-satisfying SLA. Each strategy in Figure 3 is defined by exchanges of 3 message types:

1. **Service Level Agreement Template (SLAT):** is a statement or advertisement from providers about their guaranteed service levels. Providers may deliver 1 or more SLATs representing varying service level classes. For example, Amazon EC2 offers various service levels depending on the type of AWS selected. Rackspace offers two services levels: Managed (for rapid, on-demand deployment and response) and Intensive (for highly-customized, proactive response).

2. **Service Level Objective (SLO):** is a message from a consumer defining the type and level of service they require. For example, an APP sends an SLO to a VIO declaring the size and quality of Virtual Infrastructure (VI) required for its application. Consider that Amazon EC2 users can select different sizes of EC2 instances (small, large, xlarge), each representing a different SLO.

3. **Service Level Agreement (SLA):** is an agreement that the consumer is prepared to accept. When the provider also accepts to enter the contract it is represented as **SLA***, which is equivalent to a signed SLA.

Each of the strategies are discussed and compared below. The GEYSERS SLA Management architecture and module enables each of these to be realized.

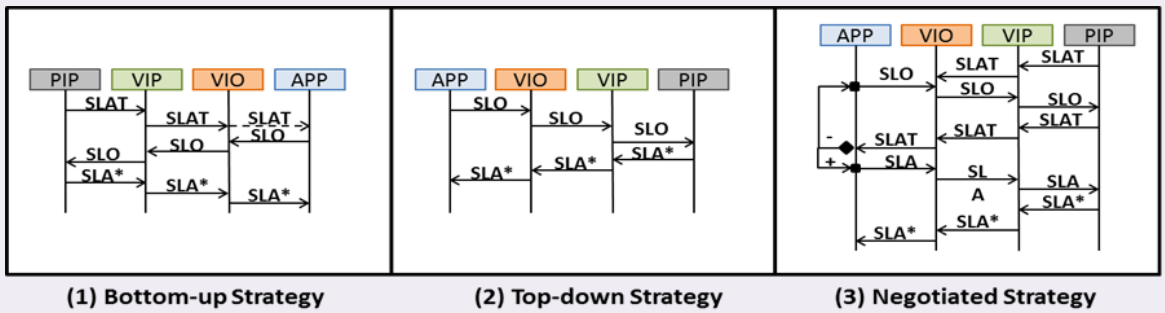

Figure 4. SLA Management strategies

"A service level management strategy is a specification of an approach or process for creating "the best" SLA given a set of SLOs and resource and service configuration possibilities."

The **Bottom-up Strategy (1)** aims at the upper layers to constantly know the service level. The main advantage is reduced risk of SLAs being compromised, as the possible SLAs are always bound by the current capabilities of the PIPs. Furthermore, the protocol is simple at all layers, without the complexity of negotiation and constant re-calculation of possible service levels at the different layers. The main disadvantage is the inflexibility of SLAs. The **Top-down Strategy (2)** occurs when a VI with specific SLA requirements is requested to the VIP by a VIO, the latter divides the VI into several sub-VI requests. This strategy is more flexible and on-demand than the bottom-up strategy. However, it moves complexity and advanced provisioning algorithms to the PIPs. There is higher risk of PIPs not being capable of responding rapidly to demands, or actually being able to resolve the mapping of VRs to physical resources. Furthermore, the VIP cannot offer guarantees to the VIO and subsequently APP with high certainty, as the lower level status is not known a priori. This is a good strategy for acquiring large volumes of small business. The **Negotiated Strategy (3)** mixes the top-down and bottom up

approach. It introduces an exchange between provider and consumer (at different levels) in order to find a service level that minimises costs for the consumer and provider while maximising their guarantees. It can be initiated by the PIP or by the APP. This strategy can support either large volumes of small business or small volumes of big business. The main disadvantage is even more complexity and overhead with negotiation. The VIPs and PIPs may spend cycles computing various mixes and options that are never used.

ARCHITECTURE AND PROFILES

The GEYSERS architecture is composed of different, autonomous roles in a layered hierarchy, each contributing differently to the lifecycle of a VI. They then have their own objectives and internal management systems. However, it is possible to generalize the specification of SLA Management base functionality regardless of the layer, although they each customize it for their purposes. These customizations are referred to as profiles. The base or reference architecture consists of 4 layers, as shown in **Figure 5**. These layers represent self-contained software bundles that can be implemented as individual client/server components or integrated with other software in the GEYSERS architecture that provides the underlying functionality required.

Figure 5. The 4 layers of the GEYSERS SLA Management architecture

"The GEYSERS architecture is composed of different, autonomous roles in a layered hierarchy, each contributing differently to the lifecycle of a VI."

The layers are described as follows:

1. **SLA Management Interface:** provides the set of operations that enable request and response interaction between the SLA Management subsystem and the outside world.
2. **SLA Logic and Persistence:** this is the core functionality that enables the SLA Management lifecycle. The logic includes rules for handling SLA-relevant events while the persistence functionality is for storing and retrieving SLA templates, agreements and monitored data.
3. **Secondary SLA Monitoring and Control:** this is monitoring and control functionality that is not directly attached to virtual or physical resources, but provides SLA-relevant events, status information and management requests using aggregation and transformation.
4. **Primary SLA Monitoring and Control:** this is the monitoring and control functionality that is directly attached to virtual or physical resources, providing raw measurements of SLA-relevant metrics.

Although these functional blocks are represented as self-contained layers, they may be distributed and replicated replicated in practice, maintaining some protocols for interaction.

Figure 6 shows more details of the computational and data components of which these layers are composed.

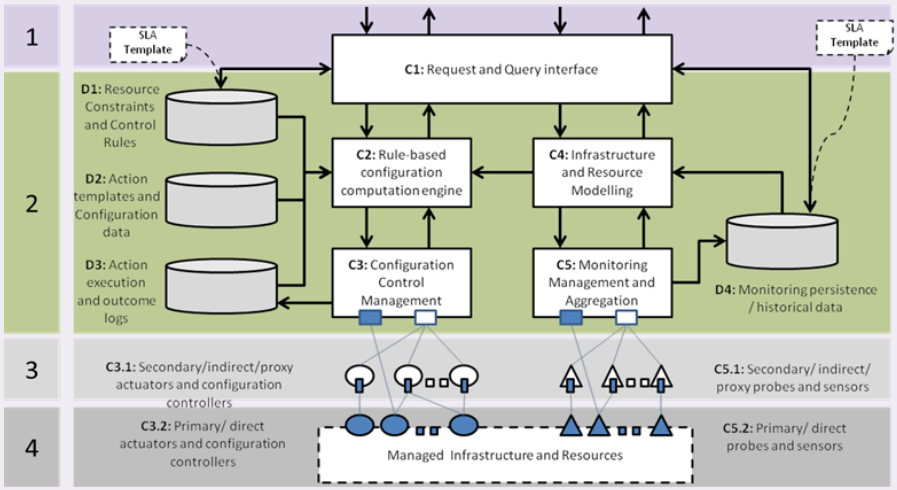

Figure 6. Detailed component architecture for GEYSERS SLA Management

"Although these functional blocks are represented as self-contained layers, they may be distributed and replicated in practice…"

Each component in the detailed architecture is provided with a label: Cx for computational components and Dx for data components. The arrows in

Figure 6 show the information flows required to implement the SLA Management lifecycle, indicating the functionality of each component. A detailed name is given to each component in the architecture, such that they are self-explanatory. Furthermore the specification of profiles clarifies their respective roles in the SLA Management architecture.

Roles	SLA Layer 1	SLA Layer 2	SLA Layer 3	SLA Layer 4
APP	Application service Templates and requests	*Out of GEYSERS scope: domain/technology specific*		
		Application lifecycle management	Application-specific sessions and metrics	Application-embedded monitors and controllers
VIO	VI service templates	VI operations and workflows	Monitoring and control of aggregate VI	VIO-specific scripts and images
VIP	VI service requests	VI level provisioning workflow management	Monitoring and control of Virtual Resources	Interface to physical resource adaptors
PIP	Infrastructure service templates and request	Physical resource adaptors and provisioning workflow management	Virtualization mechanism monitoring and control	Physical resource monitoring and control

Further details of the architecture and profiles can be found in the GEYSERS deliverable D2.6[21], where the architecture, interfaces and service provisioning workflows are refined.

LESSONS LEARNT AND OUTLOOK

The specifications of the GEYSERS SLA management concepts, template and technical architecture have gone through iterations, driven by developments in the overall GEYSERS architecture and analysis of business scenarios. It is only by thinking about the impact of the GEYSERS workflows, role interactions and virtual

[21] GEYSERS Deliverable 2.6 (31 December 2011): Refined GEYSERS architecture, interface specification and service provisioning workflow. Online: http://www.geysers.eu/images/stories/D2.6-final.pdf

infrastructure service delivery model on SLA management that the core challenges were identified and features that distinguish from the state of the art were derived. The distinction from the state of the art in SLA management has been further characterised through establishing overall challenges for realising composite IaaS. This represents a new era in the delivery and consumption of infrastructure services, where multiple resource kinds, potentially provided by multiple providers, are packaged and offered as a single service. We have addressed the challenges of SLA management for these types of services by stepping through the provisioning workflow and identifying the required templates, components and message exchanges. This resulted in a reference architecture that could be viewed from the perspective of different architectural roles, resulting in a set of profiles for the reference architecture. As we continue to develop the GEYSERS platform and build demonstrators, the core SLA management functionalities for coordinating the SLA lifecycle and managing templates are implemented as a self-contained software bundle. However, the integration with the overall GEYSERS provisioning workflows requires integration with other parts of the stack in particular the Logical Infrastructure Control Layer (LICL). SLA Management is hence not a standalone, independent function of service management – it is typically an enhancement of existing, distributed capabilities in service operations, monitoring and control. Moreover, in the case of composite IaaS, it requires collaboration across multiple service providers, while allowing them to operate in autonomy.

Editor	Philip Robinson (SAP) Philip Robinson is a Researcher in SAP's Technology Infrastructure practice, investigating novel software engineering methods and metrics to enhance and assess the manageability of Enterprise applications in Cloud Computing and Future Internet environments. In the GEYSERS project he leads the SAP team in business and requirements analysis, as well as the development of the Service Middleware Layer (SML) and Service Level Agreement (SLA) management modules.
Contributors	Alexandru-Florian Antonescu (SAP) Fabienne Anhalt (Lyatiss) José Aznar (TID) Eduard Escalona (UEssex) Joan A. García-Espín (I2CAT) Luis Miguel Contreras-Murillo (TID) Pascale Vicat-Blanc (Lyatiss)