Towards Cross Stratum SLA Management with the GEYSERS Architecture

Alexandru-Florian Antonescu[1], Philip Robinson[2]
SAP Research
[1] Zurich, Switzerland
[2] Belfast, U.K
{firstname}.{lastname}@sap.com

Sébastien Soudan, Fabienne Anhalt
Lyatiss
Lyon, France
{firstname}@lyatiss.com

Luis Miguel Contreras-Murillo, José Aznar
Telefónica I+D
Madrid, Spain
e.networkcloud@tid.es, lmcm@tid.es

Joan A. García-Espín
i2CAT Foundation
Barcelona, Spain
joan.antoni.garcia@i2cat.net

***Abstract*—**The *objectives of Cross Stratum Optimization (CSO) and SLA Management in multi-domain infrastructure services lead to challenges for maintaining autonomy of resource control and coordination of the SLA lifecycle. The GEYSERS project includes an approach and architecture for SLA Management to address these two problems. The conceptual foundations and blueprint of this architecture are presented and assessed.*

Keywords-Autonomy; Cross Stratum Optimisationl; Multiobjectives; Multi-layer Service Level Agreeement Management

I. INTRODUCTION

A Service Level Agreement (SLA) is a contract between a consumer and a provider of a service regarding its usage and quality [1][6][7]. They are important for maintaining the provider's reputation as consumers compare SLA claims and experience when selecting providers [4]. They can then plan the performance and risks of applications more optimally by having knowledge of the infrastructure's availability and performance [1][7]. For this reason providers will seek ways of ensuring optimal SLA satisfaction, within the constraints of their resource usage, energy consumption and budget constraints, in order to maintain business. Having insight or prior knowledge of application workloads and data volumes that will be running on their infrastructure is hence beneficial towards tuning their infrastructure for optimizing both of these objectives. This reflects the goals of Cross Stratum Optimisation (CSO) [7], such that an effective, automated SLA management architecture should be considered important in this context. When prioritizing operational tasks and resources based on their commitments, capacities and capabilities, providers then use the information in an SLA. Providers aim to satisfy the Service Level Objectives (SLOs) of consumers without disrupting their internal operation goals including minimisation of operational costs, power consumption, burnout of equipment and legal issues. Network operators and Internet Service Providers have been using SLAs for assuring client's experience in remote access to services and internetworking scenarios for many years [1][6]. SLAs are being used generally in Cloud Computing and Infrastructure as a Service (IaaS) for similar purposes [1][4]. The demand for rapid delivery of highly available, high performance, networked applications and infrastructure services makes automation support for SLA Management more critical.

However, the combination of CSO and SLA Management is not a straightforward, complimentary relationship, especially when delivering services composed of resources with different owners and concurrent consumers. Firstly, existing and emerging application and infrastructure services are no longer single, isolated, remote servers, virtual machines (VMs) or bandwidth. Disruptions in one resource can have consequences for the overall composite service, such that state information needs to be controllably shared. There are then dependencies across operational strata (application, machine and network) and domains with autonomous administrators, lifecycles and management controllers, which can lead to a break in the autonomy property. Secondly, application topologies continue to become more complex and heterogeneous as service-oriented design becomes de facto. For these reasons the interdependencies between application deployment, usage and network capabilities need to be considered during service planning and provisioning, making the coordination of SLA management more complex. We describe this as the *Cross Stratum (CS) SLA Management* problem.

The GEYSERS project delivers an architectural blueprint and platform for seamless and coordinated provisioning of connection-oriented networking (primarily optical) and IT (application, compute and storage) resources as infrastructure services [3]. We investigate the resultant challenges for SLA Management, which encapsulate the CS SLA Management problem introduced above, discussed more in Section II, and designed an SLA Management architecture to address the problems of autonomy and coordination. The technological requirements of CS SLA management are derived by an analysis of different strategies for multi-domain SLA management, supported by the proposed generic architecture presented in Section III. To

achieve autonomy and convergence we use an event-driven architectural style and a design that can be reused each layer of an explicit role hierarchy. This architecture is presented in Section IV. Section V discusses the value of the architecture and concludes the paper.

II. Problem Analysis and Related Work

Our analysis of the CS SLA Management problem starts with an exploration of general SLA Management challenges in a multi-provider environment as illustrated in section A. We then derive requirements for SLA Management by identifying functional blocks based on a control systems model in section B. Finally, Section C provides a brief overview of the related work.

A. The complexity of SLA management in a multiple provider environment

SLA management is initiated by a request that includes specific Service Level Objectives (SLOs). These state the consumer's desired Quality of Experience (QoE), including price, availability and response time. Beforehand each service provider creates and publishes SLA Templates (SLATs) that capture the capabilities and Quality of Service (QoS) they claim to guarantee. An SLA is hence the intersection of QoE and QoS parameters sanctioned by a consumer and provider regarding a specific service. The general SLA management problem is hence from two perspectives: consumers face the problem of determining QoE parameters that will satisfy their business demands and budget, while providers face the problem of specifying QoS parameters that are both competitive and feasible. Providers and consumers act freely within the context of resources and services they own and acquisition. Providers are free to set-up, release, and modify 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 minimization e.g. high reliability versus energy efficiency. The control objectives for applications, servers, storage and networks can conflict, as well as the SLOs of consumers and operational objectives (e.g. energy efficiency) of service providers.

In GEYSERS, physical infrastructure resources (both network and IT) are dynamically partitioned in the form of Virtual Resources (VR) and then composed into a Virtual Infrastructure (VI) to be offered to other operators as a service [3]. Examples of a VI may include links interconnecting specific ports of instantiated network VRs, or storage capacity composed by different hard disk partitions. However, the flexibility which virtualization provides from the service perspective turns on an extremely high complexity from the resource management point of view, as the virtualized physical resources can be based in very different technologies (radio, optical, electrical) that could provide the expected capabilities demanded by the final service.

Different actors maintain commercial relationships that are structured in a chain-like manner, where the different services provided by each actor are consumed by the next role in the chain, until the final service consumer. These actors are defined below (also used in the strategies in Section III):

- **Application Provider (APP):** require infrastructure services for application software and data for a set of users with certain response time objectives.
- **Virtual Infrastructure Operator (VIO):** deliver and manage a VI for APP.
- **Virtual Infrastructure Provider (VIP):** compose a VI from multiple VRs, including virtual compute, storage and network resources.
- **Physical Infrastructure Provider (PIP):** provide physical resources to host VRs.

Any service deployed using GEYSERS (e.g. connectivity service between two different data centers) is invoked with some specific characteristics, according to the SLO reference. The final user will demand consistent performance for the service supported by an SLA that will define the level of service being demanded according to a set of contractual metrics. Those metrics impose some strict obligations to the service providers, which should allocate enough resources to satisfy the SLA, while using the most adequate and efficient technology as far as the SLA is respected. Thus, the challenge is to map the SLA requirements to technology capabilities, in an efficient way and through different actors.

Furthermore, with the virtualization approach a final service could make use of resources provided by distinct operators. An obvious case could be the clear separation today among IT and network providers, but more complex combinations could also appear. This fact creates a set of interdependent providers, potentially using different technology, while ensuring the end-to-end SLA. Achieving stability and accuracy under these conditions with these complexities is hence a challenge.

B. SLA Management functional components and interactions

The functional components and interactions of SLA Management can be derived by viewing it as a class of control system, inspired by Diao et al. [2]. This is depicted in Figure 1 and explained afterwards. Some classical notations from control systems are maintained in order to reduce the ambiguity of particular information flows and component functions. Furthermore, it is intended that this model can serve as the basis for evaluating CSO and SLA Management as control theoretic problems, although this is not the intention of this paper. Subsequent work will however follow up on these principles established here. The objective is to obtain a measure of technical specificity in the architecture design.

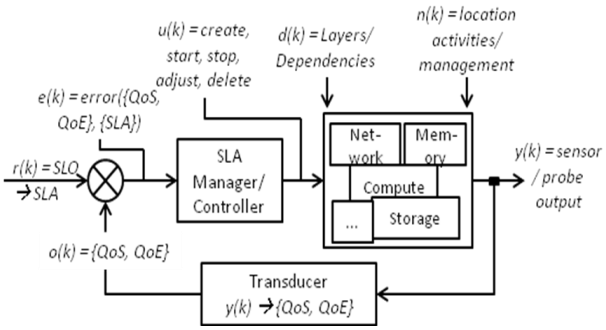

Figure 1. CS SLA Management as a control theoretic problem, where *k* represents the k^{th} entry in a control vector

The reference input *r(k)* is the set of SLOs or committed SLA parameters for the infrastructure service. For example, typical business applications have a *response time SLO of 2s per request* as a QoE stipulation. Considering the following top → down layers for a hosted application stack (being (6) *Application*→ (5) *Service* → (4) *Virtual Machine* → (3) *Virtual Network* → (2) *Physical Host* → (1) *Physical network*) the bounds of the SLO are progressively reduced, such that $r_6(k = 2) > r_5(k) > \ldots > r_1(k)$. As the infrastructure service is in operation, a set of measured output *y(k)* is returned from sensors and probes attached to the managed resources (as the example deals with response time, probes for network nodes, memory, compute and storage I/O are relevant). The output *y(k)* is transformed to a QoS and QoE metrics *o(k)* in an online or offline manner, such that they can be compared with the parameters in the reference SLO/SLA. This comparison results in an error calculation *e(k)*, which informs the SLA Manager, and controller of the infrastructure service, about any error resolution actions that need to be taken. These actions are the control input *u(k)* used to adjust the target infrastructure service. These actions include create/delete/adjust and start/stop of resources and their tunable parameters. As the hierarchy of layers increases and dependencies between resource types increase, there is a propagating effect from the execution of control actions. For example, on the IT side stopping a physical host causes all virtual machines and host to be unavailable. This impact is known as the disturbance *d(k)*. Measurements at the target are also affected by other noise *n(k)* (e.g. time delays in exchanging events between layers and potential overhead of the management system itself) that can place some imprecision on the readings. In the case of infrastructure as a service and virtualization, it is likely that multiple workloads will be contending for the same physical resources.

C. Related Work

Zhang and Song [5] propose an architecture for SLA Management taking into account the five phases of SLA lifecycle. They also consider different roles (service provider, service broker, SLA Manager, service client) and implement this via two modules (offline and runtime). The offline module handles SLA registrations and client queries, template storage. The runtime module handles monitoring, logging and controlling (detects SLA violations, possible admission control). As their proposed architecture is single-domain oriented its implementation is monolithic and there are no events utilized. In our proposed architecture feedback is incorporated from more than one source (e.g. different providers and clients) taken into consideration the distributed nature of the architecture.

Theilmann, Happe et al. [7] propose an architecture for hierarchical SLA Management in service oriented environments by considering the entire lifecycle of SLAs and services. Their architecture is also event-based and utilizes two types of models: SLAs for the communication within and among SLA Managers, as well with external providers and a model for describing creation of new service instances based on SLAs. Particular to this architecture is its adaptation to software service environments and not infrastructure services. Furthermore they assume exclusive control of the available resources.

III. Multi-Domain SLA Management Strategies

We introduce three strategies for multi-domain SLA Management, each towards supporting CSO. Each strategy provides a protocol for hierarchical controllers (or controller domains) to pass reference inputs and measured output between each other. These controller domains are the PIP, VIP, VIO and APP actors introduced in Section II.A.

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 combination of 1 and 2 with additional message exchanges.

A. Bottom-Up Strategy

This strategy is initiated by PIPs advertising their capabilities to one or more VIPs. The objective of the strategy is for the upper layers to constantly know what the service level is that can be supplied at any time. Upper layers restrict their demands to the known service levels that will be available. Explicitly stated, application providers (APPs) only request applications from a specific VIO within the application service levels stated by the VIO. Subsequently, the application service levels are bound by VI service levels registered at the VIO, which are bound by VR service levels registered at the VIP, which are bound by resource capabilities of the PIPs. The protocol (stages s0 to s7) for this strategy is depicted in and described below.

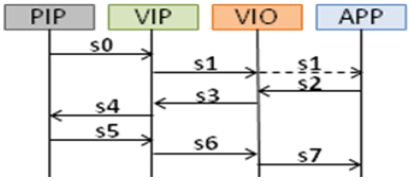

Figure 2. Bottom-up strategy for SLA management

s0. Each PIP computes and declares their resource capabilities to a VIP in the form of Virtual Resource (VR) service levels.
s1. The VIP uses the VR service levels to compute a set of VI service levels that it can realistically offer to VIOs. Application Providers (APP) may also receive this information via the VIO.
s2. Once the APP knows the details of their application demands, they request usage of the VI operated by the VIO with a specific service level. If the APP demand does not fit a predefined service level, it is refused.
s3. The VIO makes a request to the VIP to provision a VI for the APP at the required service level.
s4. The VIP sends VR provisioning request to each of the PIPs associated at the required service level.
s5. Once the VRs are provisioned and reserved for the specific VI, a response and indicator is sent to the VIP representing a "signed SLA" between the PIP and VIP.
s6. Once the VIP has all VRs provisioned, the VIO is given a notification that the VI is ready at the required service level => "signed SLA" between the VIP and VIO.
s7. The VIO can then send a response or refuse (in the case where a predefined service level was not selected or is no longer available) to the APP.

B. Top-Down Strategy

In this strategy, 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. The sub-VI represents a subset of the VRs of the original VI request, each VR having several SLA requirements. The VIP then sends each sub-VI request to a PIP, and asks if all the VRs of the sub-VI can be allocated on his physical resources and if the SLAs can be satisfied. Hence it is up to the PIP to find a mapping solution for allocating the VRs on the physical resources such that the SLA for each VR is satisfied. If no mapping solution can be found, the VI request is rejected, and the VIP needs to change the division into sub-VIs and ask again an allocation.

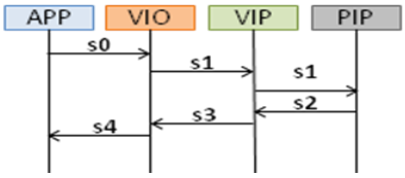

Figure 3. Top-down strategy for SLA management

s0. An APP may initiate this strategy by defining a demand or VI support request. Unlike the bottom-up strategy, there is no assumption that the APP knows the bounds on possible service levels; these service levels from lower levels become apparent to upper layers on demand.
s1. The VIO sends a VI request to the VIP with specific service level objectives. The VI creates multiple sub-VI requests, which can be associated with individual VR requests.
s2. Each VR request is then sent to the respective PIPs to request them to allocate resources for provisioning the VRs to satisfy the individual service level objectives.
s3. Each PIP is responsible for mapping the VRs to physical resources and executing the allocation algorithm to satisfy the VR's service level objectives. The completion of this mapping and response to the VIP represents a signed SLA between a PIP and VIP.
s4. The VIP returns a positive result to the VIO if all service level objectives per sub-VI VR can be allocated and connected amongst the set of PIPs. This represents a signed SLA between the VIP and VIO. Otherwise a refusal is returned.
s5. Similarly, the VIO responds with a signed SLA with the APP or a refusal, depending on the outcome of the mapping at the lower levels, and VI service offered by the VIP.

C. Mixed/ Negotiated Strategy

This third strategy 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 (here the PIP is the initiator). The VIO and VIP will also constantly update and advertise their VI service levels stored locally. The objective is hence to find a "right-sized" match between demand, service level and resource provisioning, without over or under-estimating what the PIPs can deliver. The exchange between the providers and consumers at different levels is known as a negotiation protocol.

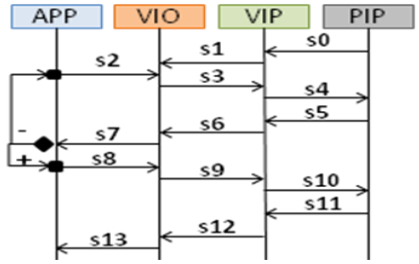

Figure 4. Negotiated strategy for SLA management

s0. In the case a PIP acts as initiator, each PIP computes and submits their current resource capabilities as VR service levels to the VIP.
s1. The VIP computes the set of VI levels that it can realistically offer, given the set of VR service levels from the PIPs.
s2. An APP could also initiate the protocol. The APP sends a VI service request to the VIO
s3. The VIO checks which of its locally-registered service levels are closest to the APP's VI service level request. It sends a request to the VIP with this VI SLO request.

s4. The VIP does a similar check for the closest, known VR service levels for the request and sends an updated request to the PIPs

s5. Each PIP may do a readjustment and reallocation of VRs to physical resources to better tune the guarantees of the VR to the required service level. However, they will also want to consider their internal operational objectives. The response to the VIP is then an "offer", as opposed to a "confirmation" in the other strategies.

s6. The VIP issues an offer of a VI service level to the VIO, which should be closer (or the best approximation) to the level of the initial request.

s7. The VIO creates a service level offer to the APP. The APP may refuse this (-) and resend a request (s2), or the APP may accept this (+), such that the protocol moves to the next stage.

s8. If APP accepts, the rest of the protocol follows the top-down strategy messaging, with the exception that the PIP already has the planning done for the service level.

s9. The VIO sends a VI request to the VIP with specific SLOs. The VI creates multiple sub-VI requests, which can be associated with individual VR requests.

s10. Each VR request is then sent to the respective PIPs as in the top-down strategy to request them to allocate resources for provisioning the VRs to satisfy the individual service level objectives.

s11. Each PIP is responsible for mapping the VRs to physical resources and executing the allocation algorithm to satisfy the VR's service level objectives. The completion of this mapping and response to the VIP represents a signed SLA between a PIP and VIP.

s12. The VIP returns a positive result to the VIO if all service level objectives per sub-VI VR can be allocated and connected amongst the set of PIPs. This represents a signed SLA between the VIP and VIO.

s13. Similarly, the VIO responds with a signed SLA with the APP or a refusal, depending on the outcome of the mapping at the lower levels, and VI service offered by the VIP.

D. Strategies comparison

The main advantage of the bottom-up strategy is that there 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 PIPs do not need to be very sophisticated with regards to the allocation algorithms, such that the provisioning time for a VI is low. The main disadvantage is the inflexibility of SLAs and the increased likelihood that new business will be turned away. Additionally, consumers may have to select service levels above or below their true demand, resulting in either over or under provisioning of resources. This is however a good strategy when the scope of demands from the APP are sufficiently well known at each level of the architecture.

In the case of the top-down strategy, it 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.

Finally, the main advantage of the mixed strategy is that the best of both previous strategies is inherited. 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.

IV. GEYSERS SLA MANAGEMENT ARCHITECTURE

The GEYSERS SLA Management architecture is designed to implement the three strategies presented in the previous section. However, in the reference implementation of the architecture, the VIP decides which strategy to use, based on the requirements of the virtual infrastructure. The current practices in SLA handling [9][10] mostly resemble the bottom-up strategy in which the providers advertise their own capabilities and guarantees, constraining the choice for the end user to accepting the offered service level. Our approach allows more freedom for the end user in defining more fine-grained requirements for the infrastructure support its own applications.

We use a hierarchical event-driven architecture style as our approach. This approach supports *autonomy of providers and management domains*, such that they can implement their own policies and operational objectives for their resources. This hierarchical, event-driven architecture style further supports *collaboration* and *coordination* between providers and management domains in uniformly exchanging relevant state information and handling dependencies between resource types (i.e. service access, network, compute, storage).

A. Overall Architecture

The architecture corresponds to the control systems model in Figure 1. There are five components in the architecture shown in Figure 5, including the correspondence to the control systems model.

The *Request Handling* component receives service requests with SLOs from upper layers as reference input $r_{l+1}(k)$, where l is the local layer and l+1 represents the upper layer. The request is converted to a local input reference r(k) that is used to initialize the SLA control system. Secondly, the *Control* component generates control input for local resources based on an error evaluation e(k) = eval (r(k), $r(k_{t-1})$, $o(k_t)$, $o(k_{t-1})$), where r(k) is the current SLO, $r(k_{t-1})$ is historic SLOs/ reference inputs for the same target and metric, $o(k_t)$ = o(k) is the output or feedback about the

metric k and $o(k_{t-1})$ is historic data about the metric k. If resources to be changed are external, then Outgoing Requests are generated ($u(k) \rightarrow r_{l-1}(k)$) and communicated to the respective provider.

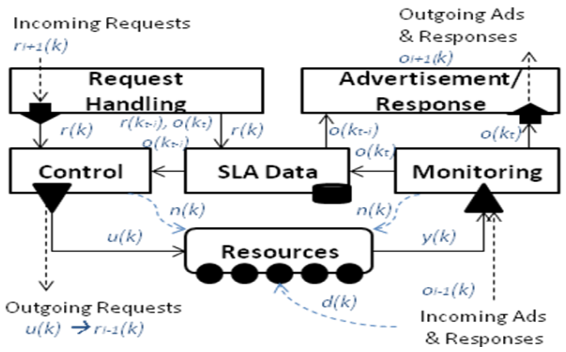

Figure 5. GEYSERS SLA Management Architecture with annotations to show correspondence to Figure 1

There is some noise $n_{ctr}(k)$ introduced. The *Monitoring* component then gathers, filters and transforms measured output y(k) from local resources, as well incoming advertisements and responses from lower layers ($o_{l-1}(k) \rightarrow y_{l-1}(k)$). It also issues advertisements and feedback about the state of its local resources $o(k_t)$, and updates the status history of the local *SLA Data* component, which maintains a history of output $o(k_t)$ and SLOs $r(k_t)$ over a time period $T = (t_{n-m} - t_n)$. Finally, the *Advertisement and Response* component sends feedback $o_{l+1}(k)$ to an upper layer that requires notification of its resource states.

B. *Vertical and Horizontal Architecture Profiles*

Although we refer to upper and lower layers in the previous section, the architecture is actually designed to flexibly support vertical and horizontal operation. Horizontal interaction is seen here between a PIP providing IT (compute and storage) resources and another providing the optical network resources. The events used for interaction are Requests, Responses and Advertisements (Ads) as shown in Figure 6.

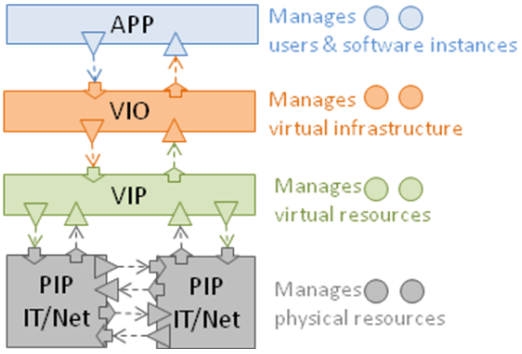

Figure 6. GEYSERS Layered SLA Management

We assume that each event includes its source, type, timestamp and a payload that updates the value of a metric k. Figure 6 shows that each level of the GEYSERS architecture implements the general SLA Management architecture but has different classes of resources to be managed. Different classes of resources indicate different classes of control actions, probes and SLA data.

C. *Analysis of the Architecture*

As an analysis we step through a simple scenario. Consider the APP has a response-time objective of k = 1.0s sent as a request to the VIO. The VIO then calculates $r_{VIO}(k) < r_{APP}(k)$ for the response time of the virtual infrastructure. The VIP further transforms this to a network bandwidth requirement, assuming knowledge of the estimated volume, payload and number of concurrent users associated with the APP's foreseen usage. This bandwidth requirement $r_{PIP}(k)$ is passed down to the PIP without the PIP having to know about the APP's initial response time SLO. As the resources are provisioned at the PIPs, they send updates to the VIP, which is then able to assert the availability of VRs. Subsequently, the VIP also receives notifications about the assembly of the VI. Alternatively, using a bottom-up strategy, the PIPs could have issued advertisements to the VIP about its physical resource capabilities using the same request-response event model. This demonstrates that both the top-down and bottom-up strategies could be supported with this architecture. Moreover, the event approach supports the CSO objectives of keeping PIPs aware of application demands, while informing application providers about the relevant state of the infrastructure.

V. Outlook and Conclusions

In developing the GEYSERS SLA Management architecture, we have derived requirements, a reference model and evaluation criteria for addressing the Cross Stratum SLA Management problem – supporting CSO without removing provider autonomy and enhancing collaboration. The approach of describing SLA Management as a control theoretic problem aids in defining concrete objectives for CS SLA Management. We introduced and compared three alternative strategies for CS SLA Management as protocols, discussing how variations in the operational context change their applicability. We then provided the outline of an architecture that addresses the CS SLA Management problem and can be used for all three strategies. This architecture continues to be developed in the GEYSERS project, where we see the most potential for addressing SLA Management in distributed Cloud Computing services connected by (optical) network providers.

From our initial implementation, we approximated the effort required for adding a new PIP to the system. In order to do this, the provider needs to implement a resource adapter which will be used for both resource discovery and operating and monitoring the physical resources. The implementation effort depends on the complexity of the domain controller and on the structure of the inputs and outputs used by the system. In the case that the new provider

offers a new kind of resource (different than the OCCI [11] resource kinds), this would imply changing GEYSERS internal information model and adding new operations for that new resource kind. Even in this case, given the fact that the GEYSERS framework uses generic operations of instantiation, configuration, operational and decommission, it is possible to write adapter logic with a few lines of code to map them on to the actual physical operations. As an example, GEYSERS implementation contains four such physical resource adapters for various computing and network system managers. An implementation based on a popular, open-source cloud manager contained a maximum of 256 lines of code in a method and 281 statements overall, representing a small programming effort.

ACKNOWLEDGMENT

The work in this paper has been (partly) funded by the European Union through project GEYSERS (contract no. FP7-ICT-248657).

REFERENCES

[1] P. Patel, A. Ranabahu and A. Sheth, "Service Level Agreement in Cloud Computing." Cloud Workshops at OOPSLA09, 1-10, 2009

[2] Diao, Y. D. Y., Hellerstein, J. L., Parekh, S., Griffith, R., Kaiser, G. E., & Phung, D. "A control theory foundation for self-managing computing systems." IEEE Journal on Selected Areas in Communications, 2005.

[3] E. Escalona , S. Peng, R. Nejabati, D. Simeonidou, J.A. Garcia-Espin, 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

[4] L. Wu and R. Buyya, "Service Level Agreement (SLA) in Utility Computing Systems" In V. Cardellini, E. Casalicchio, K. Castelo Branco, J. Estrella, & F. Monaco (Eds.), Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions (pp. 1-25), 2011

[5] S. Zhang and M. Song, "An architecture design of life cycle based SLA management," Advanced Communication Technology (ICACT), 2010 The 12th International Conference on , vol.2, no., pp.1351-1355, 7-10 Feb. 2010

[6] TMForum's SLA Management website, Online: http://www.tmforum.org/SLAManagement/1690/home.html (last accessed 20/20/2012)

[7] W. Theilmann, J. Happe, C. Kotsokalis, A. Edmonds, K. Kearney, J. Lambe, ":A Reference Architecture for Multi-Level SLA Management," Journal of Internet Engineering, Vol. 4, No. 1, December 2010

[8] Y.Lee et al., Research Proposal for Cross Stratum Optimization (CSO) between Data Centers and Networks", IETF Internet Draft, Available Online: http://tools.ietf.org/html/draft-lee-cross-stratum-optimization-datacenter-00, (Last Accessed 20.02.2012)

[9] Amazon EC2 Service Level Agreement, online: http://aws.amazon.com/ec2-sla/ (Last Accessed: 22.02.2012)

[10] Microsoft Azure Support: Service Level Agreements, online: http://www.windowsazure.com/en-us/support/sla/ (Last Accessed 22.02.2012)

[11] Open Cloud Computing Interface, online: http://occi-wg.org/ (Last Accessed 21.02.2012)