Towards an Autonomic Service Architecture

Ramy Farha[1], Myung Sup Kim[1],
Alberto Leon-Garcia[1], and James Won-Ki Hong[2]

[1] Dept. of Electrical and Computer Eng'g, University of Toronto, Toronto, Canada
{ramy.farha,myungsup.kim,alberto.leongarcia}@utoronto.ca
[2] Dept. of Computer Science and Eng'g, POSTECH, Pohang, Korea
jwkhong@postech.ac.kr

Abstract. Traditional telecommunications service providers are undergoing a transition to a shared infrastructure in which multiple services will be offered to customers. These services will be introduced, modified, and retired at a pace that tracks changing requirements and demands. In order to be cost-effective, these services will need to be delivered over a shared infrastructure that is managed to support delivery requirements at a given point in time. In this paper, we present an Autonomic Service Architecture (ASA) for the automated management of networking and computing resources. ASA ensures the delivery of services according to specific agreements between customers and service providers.

1 Introduction

The transition from circuit-based telephone networks to IP-based packet networks presages the replacement of traditional voice telephony service by a broad array of media-rich, personalized and context-aware services that need to provide immediacy, reliability, and consistency of quality levels. Service providers (SPs) will need to be able to deploy, maintain, and retire services quickly according to demand. This future service environment requires a new service delivery framework that can exploit the capabilities of IP networks and provide required service richness, agility, and flexibility. From the information technology (IT) world, autonomic computing [1] is touted as the means to providing a rich set of IT services over a common computing infrastructure. A key feature of autonomic computing is the automated management of computing resources. The application of autonomic management principles to ensure the delivery of telecommunications services is largely unexplored. In this paper, we introduce an Autonomic Service Architecture (ASA) to address this need.

As mentioned before, the first work in the autonomics wave has been the Autonomic Computing proposal by IBM. However, IBM's approach exclusively focuses on computing resources for IT services delivery. Our work aims to expand this basic view to include telecommunications services, which consist of both computing and networking resources. Another proposal called Autonomic Communication [2] has similar aims to IBM's Autonomic Computing, except that it focuses on individual network elements, and studies how the desired element's behavior is learned, influenced or changed, and how it affects other elements. Our work is focused on services, and therefore is a top-down approach as compared to this bottom-up approach. In addition, research projects such as Autonomia [3], AutoMate [4], and Oceano [5] are using the autonomic concept in various ways. Autonomia provides dynamically programmable control and management to support development and deployment of smart applications. AutoMate enables development of autonomic Grid applications that are context-aware, self-configuring, self-composing, and self-optimizing. Oceano is developing a prototype for a scaleable infrastructure to enable multi-enterprise hosting on a virtualized collection of hardware resources. The closest work to ours is that of HP [6], which proposes an architecture and some algorithms for a service-oriented control system that constantly re-evaluates system conditions and re-adjusts service placements and capacities. The control system is organized as an overlay topology with monitoring and actuation interfaces to underlying services. The difference between our work and these approaches is that they consider specific services to which their design is appropriate. We propose a generic architecture which applies to all services.

The main contribution of our paper is that it proposes ASA as a generic architecture for autonomic service delivery by SPs. ASA is service-independent, and defines a resource management model based on virtualization. The rest of this paper is structured as follows. Section 2 describes ASA. Section 3 illustrates the operation of the Autonomic Resource Broker (ARB), the key component of ASA. Section 4 concludes this paper.

2 Autonomic Service Architecture

The players involved in the delivery of a service are the customers and the SPs. We define a service as the engagement of resources for a period of time according to a contractual relationship between the customers and the SPs. Resources are physical and logical components used to construct services. Service Level Agreements (SLAs) are contracts between SPs and customers [7], which are critical in guaranteeing service delivery. Service management ensures that SLAs are met, and that the necessary resources for the service delivery are provided.

After customers and SPs negotiate the SLA, ASA will manage the service in order to meet SLAs without SP's intervention. In this section, we will present ASA according to a layered view, where services are built on underlying virtual and physical resources. If the problems incurred are too complex to be handled autonomically, manual adjustments are needed. When customers purchase a service from a SP, they can themselves offer this service to other customers, becoming SPs to those customers. ASA ensures End-to-End (E2E) service delivery, where SPs negotiate with other SPs involved in the E2E delivery, to which they become customers.

The autonomic service architecture (ASA) is driven by our view that "everything is a service". We identify two types of services, however: Basic services, which cannot be broken down anymore into other services, and which mainly consist of the underlying resources, and Composite services, which are composed from basic services as well as from other composite services.

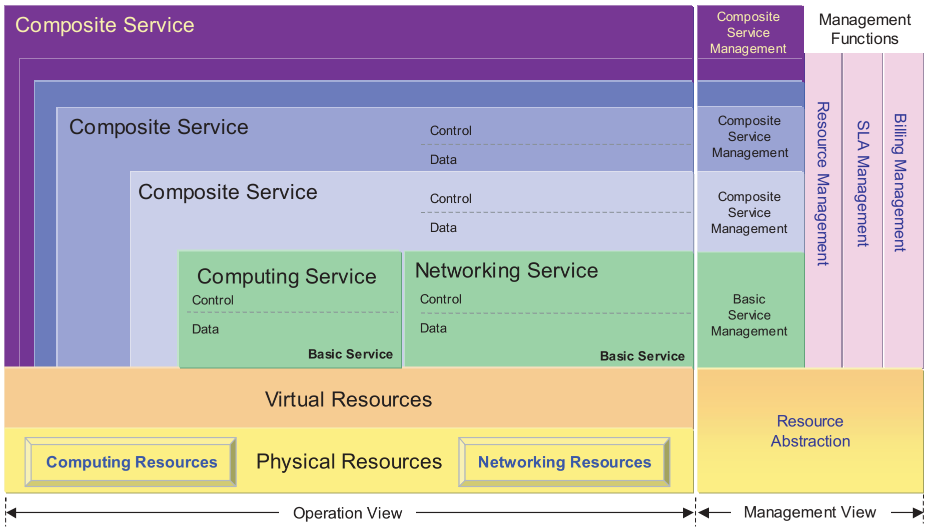

Fig. 1. Autonomic Service Architecture Layered View

Fig. 1 shows ASA's layered view of services. The lowest layer consists of physical resources engaged in the delivery of the service. The middle layer consists of an abstraction of the physical resources into virtual resources according to metrics specifying the characteristics of the physical resources. The upper layers consist of services composed using these underlying resources. Vertically, services are broken into two views: Operation and Management. The Operation view consists of the control and data planes at different layers, while the Management view consists of the management structure needed to manage these services.

2.1 Operation View

The operation view consists of a layering of resources.

Physical Resources Layer. This layer consists of the physical resources that the SP has at its disposal. These resources are either computing (Servers, Workstations, Storage, Clusters), and/or networking (Routers, Switches, Links).

Virtual Resources Layer. This layer abstracts physical resources into virtual resources. Virtualization allows the SP to deal with resources at its disposal to create services *independent* of the actual physical resources. Every service views its needed resources as an array of metrics from a unified language we will define, called the Common Resource Format (CRF). The motivation for CRF is similar to the motivation that drove IBM to propose the Common Base Event (CBE) model [8] to translate proprietary application logs into standard CBE format. From physical to virtual resources, translation is needed from proprietary formats to CRF using an adapter. Virtualization depends on the types of resources involved. Fig. 2 shows three different types of virtual resources:

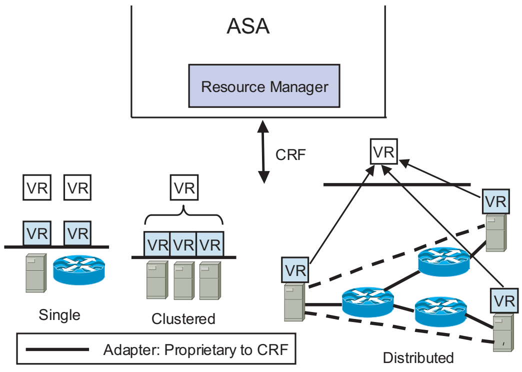

Fig. 2. Virtualization of Resources

1 **Single resources:** Consist of a single physical resource (router, server).
2 **Clustered resources:** Consist of multiple physical resources at a certain geographical location (cluster).
3 **Distributed resources:** Consist of multiple physical resources, geographically dispersed, virtualized to look as an aggregate resource (grid [9]).

Basic Services Layer. Some virtual resources, such as Virtual Networks [10], can be offered directly to customers by SPs as basic networking services, consisting of guaranteed IP transport. In other situations, basic services, which are bought from other SPs according to SLAs, become virtual resources at the disposal of the purchasing SP's composite services.

Composite Services Layer. Composite services consist of several basic services and/or composite services. The composite services can be offered directly to customers. Otherwise, the composite services are virtual resources to other composite services. The process of service composition is hierarchical and recursive, and continues until the composite service is offered to customers.

2.2 Management View

The main task of ASA consists of managing the resources available to the SP in order to meet changes in service demands and user requirements. All management functions (Resource, SLA, Billing) are performed by the Autonomic Resource Broker (ARB), a self-managing entity whose role is to ensure automated delivery of services. A key concept in the IBM autonomic computing architecture is the autonomic element, a component that is responsible for managing its own behavior in accordance with high-level policies, and for interacting with other autonomic elements. In ASA, the Autonomic Resource Brokers (ARBs) are the analogy of autonomic elements. We define the ARB as the component responsible for managing a service instance in accordance with policies, by interacting with underlying resources, and with other ARBs to provide or consume services. ARBs follow policies to ensure SLAs between customers and SPs are met.

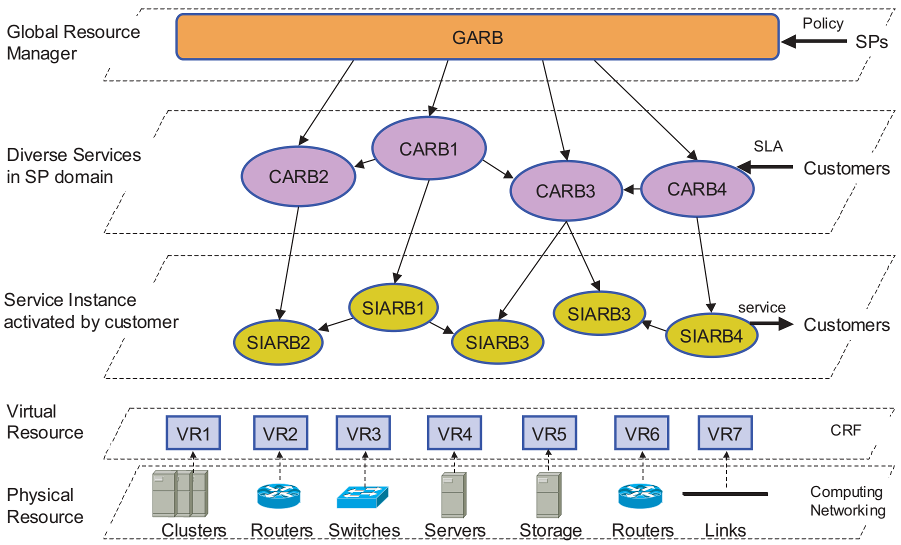

Fig. 3. Hierarchical Management View

When customers activate service instances, these instances are managed by SIARBs (Service Instance ARBs). The multiple service instances of a particular service offered by a SP are managed by CARBs (Composite ARBs). Some CARBs, called basic CARBs, manage the basic services consisting of virtual resources. Other CARBs, called composite CARBs, manage composite services. The different services offered by a SP are managed by a GARB (Global ARB), which handles all the resources available at this SP's disposal. The hierarchical structure of ARBs is shown in Fig. 3. Note that ASA is based on service-oriented architectures [11] for interactions between ARBs, and with underlying resources.

3 Autonomic Resource Broker Architecture

These ARBs are self-managed according to high-level policies. ARBs handle the autonomic operation of ASA. Fig. 4 shows the ARB's internal architecture and the flow of information between the different ARB components.

3.1 Information Bases

Information Bases store the information needed for ASA to autonomically deliver services, in a self-configuring, self-optimizing, self-healing, and self-protecting way. Information Bases can be classified into five logical groupings:

Customer Information Base (CIB): Contains information about customers, such as personal data, list of services subscribed to with their SLA, and bill.

Service Information Base (SIB): Contains information about the service instances activated by the customers, such as parties involved (customer and

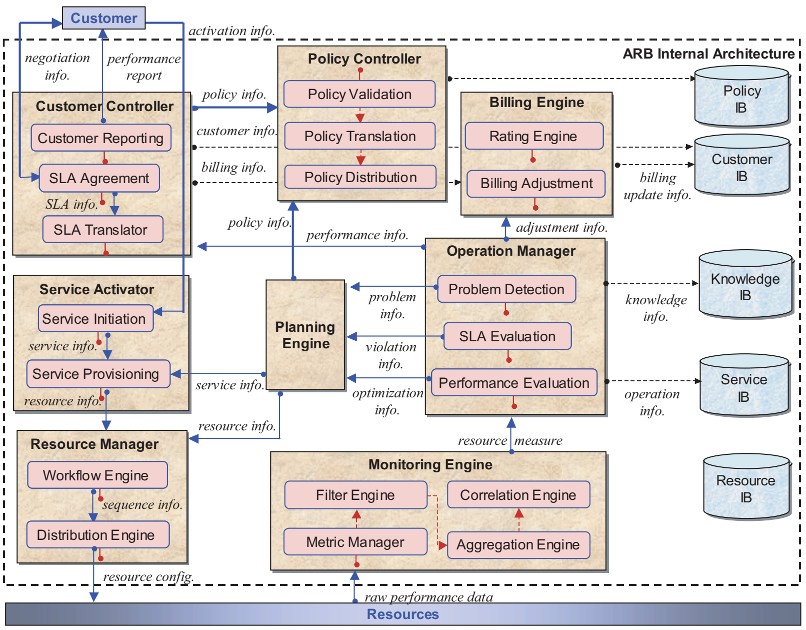

Fig. 4. Autonomic Resource Broker Architecture

SP), SLA agreed upon, types of resources needed, amount of each resource type needed, billing plan, and operation history.

Resource Information Base (RIB): Contains information about the resources available, such as types of resources and quantity.

Policy Information Base (PIB): Contains policies created at runtime, or entered manually. These policies are service-based. There are policies regulating the operation of each ARB component, as well as providing SLA Templates for services offered and specifying types of resources needed for services offered.

Knowledge Information Base (KIB): Contains information for use in case problems arise and remedy actions can be taken based on a previous occurrence of the problem, such as problem description, problem cause, time of occurrence, parties involved, elaborated solutions, and effect of solutions.

3.2 Policy Controller

Policies are created initially by SPs, or at runtime as a result of customers activating services. Existing policies are updated as a result of service demand and load variations. The Policy Controller entails the following actions:

Policy Validation. The creation or update of policies could lead to conflicts, redundancy, inconsistency, and infeasibility. This component ensures no such problems occur and remedies to them.

Policy Translation. This component interprets policies and translates them to an understandable format for use by ARB components.

Policy Distribution. This component distributes policies to ARB components, to the PIB, or to other ARBs.

3.3 Customer Controller

This component constitutes the only interface between the customers and the SPs. This is part of the self-configuring aspect of ASA. Note that it is essential to differentiate between the SLA Agreement and the Service Initiation procedures. The SLA Agreement is a one-time operation that occurs when customers buy the service from SPs. Of course, the customer, using this same procedure, could later modify the service bought, but this operation does not occur for each service instance activation by the customer, as the Service Initiation does. The Customer Controller entails the following actions:

Customer Reporting. Customers have access to some monitoring results, in order to allow them to switch SPs if performance is not satisfactory.

SLA Agreement. The SLA is negotiated between customers and SPs, following a negotiation protocol we define.

SLA Translator. Once the SLA negotiation between the customer and SPs is completed, the SLA Translator creates/updates policies on the fly to regulate the purchased service's delivery. In addition, the SP fills the CIB with customer and billing information. Note that the SLA Translator itself is policy-based, i.e. its operations are regulated by manual policies entered by the SPs which guide its operations, based on services involved, customer types, or other criteria.

3.4 Service Activator

This component is invoked upon service activation by customers. This is part of the self-configuring and self-optimizing aspects of ASA. The Service Activator entails the following actions:

Service Initiation. Customers activate services they already bought. Information, related to the service activated and to the customers, is retrieved and used for Service Provisioning.

Service Provisioning. As mentioned previously, ASA was built based upon the basic premise that "everything is a service". Using this approach to services, composite services are composed out of basic services and of other composite services, and provisioning composite services consists of choosing the appropriate amounts of resources to allocate to each service component. The basic services needed are identified, and the objective function for this service is used to optimize provisioning. The amount of resources needed of each type are calculated, and this information is sent to the Resource Manager.

3.5 Resource Manager

This component allocates/provisions resources available as needed by the service instance that was activated. The appropriate ARBs and/or the underlying resources have to be contacted. This is part of the self-configuring and self-optimizing aspects of ASA. The Resource Manager entails the following actions:

Workflow Engine. The resource allocation process is converted to a workflow of actions executed on underlying resources and/or appropriate ARBs.

Distribution Engine. The actions decided by the Workflow Engine are distributed to underlying resources and/or appropriate ARBs.

3.6 Monitoring Engine

This component monitors the raw performance data sent from underlying resources and/or appropriate ARBs. This is part of the self-healing and self-protecting aspects of ASA. The Monitoring Engine entails the following actions:

Metric Manager. There is a need to quantify raw performance data in a common format (CRF) understandable by ARB components to make decisions.

Filter Engine. The mechanisms to filter unwanted data for all ARBs need to be specified. At the ARB components, the tradeoff is between precision and overhead of measurements. The more precise the results need to be, the more measurements we need to perform.

Aggregation Engine. The measurements received after filtering could be aggregated if a new metric such as a summation, average, maximum, or minimum of the measurements collected is needed.

Correlation Engine. The filtered and aggregated measurements are correlated and complex situations are detected, using techniques such as spatial/temporal correlation, and prediction.

3.7 Operation Manager

This component analyzes ARB operations, and detects any abnormal behavior that results from faults, SLA violation, or sub-optimal performance. This is part of the self-optimizing, self-healing, and self-protecting aspects of ASA. The Operation Manager entails the following actions:

Problem Detection. Faults occur when computing or networking components fail. Overloads occur when the demand on a component exceeds the capacity of that component. Congestions occur when the performance of some components degrades due to excessive load. If problems are detected, the Planning Engine is notified. This is part of the self-healing aspect of ASA.

SLA Evaluation. SLAs are evaluated, and violations detected are sent to the Planning Engine, and to the Billing Engine for proper adjustments to the customer bill in the CIB. This is part of the self-optimizing aspect of ASA.

Performance Evaluation. When the operation is satisfactory (no problems or violations), ARB ensures resources are optimally allocated, and if not, notifies the Planning Engine. This is part of the self-optimizing aspect of ASA.

3.8 Planning Engine

This component is considered to be the brain of the ARB. This is part of the self-optimizing, self-protecting, and self-healing aspects of ASA. The inputs to the Planning Engine are:

– Customer entry, i.e. the customer information related to the services where problems have occurred, and their SLAs. These can be obtained from the Customer Information Base (CIB).
– Service entry, i.e. performance requirements for services (SLAs). These can be obtained from the Service Information Base (SIB).
– Policy entry, i.e. policies that restrict allocation of resources and constrain solutions. These can be obtained from the Policy Information Base (PIB).
– Resource entry, i.e. resources at the SP's disposal extracted from the resource pool. These can be obtained from the Resource Information Base (RIB).
– Knowledge entry, i.e. previous comparable situations, where the advocated solutions could be used instead of elaborating new ones. These can be obtained from the Knowledge Information Base (KIB).
– Notifications from the Operation Manager component to indicate problems detected, SLA violations, and sub-optimal performance.

The outputs that can be generated by the Planning Engine are:

– Changes to the Service Provisioning, for instance resources needed to meet service requirements, re-allocation plans to improve service performance.
– Changes to the Resource Manager when the problems are not drastic enough to require re-provisioning of the service instance.
– Changes to the policies regulating the operation of ARB components.

3.9 Billing Engine

This component bills customers using three charges: A flat charge, a usage charge, and a content charge. This allows more flexibility in using the appropriate billing solution appropriate for each service, class of service, or even individual customer. The pricing is adjusted based on congestion and on billing policies to reflect SLA violations, and to allow pricing to be used as a congestion control technique by increasing the prices of the resources when the network is congested. Note that the exact details of the billing plan are determined when the customer buys the service. This is part of the self-healing aspect of ASA.

4 Conclusion

The journey to a fully autonomic service architecture is still in its early stages. This paper illustrates our proposed generic approach towards this goal, using ASA. ASA allows service providers to reduce service delivery costs to customers. ASA is based on two main concepts: virtualization of physical resources using a common language (CRF), and autonomic service delivery using a hierarchy of Autonomic Resource Brokers (ARBs). The hierarchical service view allows ASA to easily expand to next-generation services by allowing flexible, scalable, and recursive service management. ASA is still a conceptual architecture, but its realization for real services is underway. First, we are defining XML formats for the information bases, for policies, for service and SLA templates, as well as elaborating CRF. Second, we are defining interfaces among ARBs, exploring several ARB topologies (Peer-to-Peer, Hierarchical, Hybrid) and assessing them. Third, we are developing algorithms for each ARB functional block (e.g. Service Activator, Operation Manager, Planning Engine etc.). Finally, we are implementing ASA in our Network Architecture Laboratory at the University of Toronto, for specific services such as Voice over IP (VoIP).

References

1. IBM Corporation: An architectural blueprint for autonomic computing. White Paper, (2003)
2. Autonomic Communication: http://www.autonomic-communication.org
3. Xiangdong, D. et. al.: Autonomia: an autonomic computing environment. Proceedings of the IEEE International Performance, Computing, and Communications Conference (2003) 61–68
4. Agarwal, M. et. al.: AutoMate: enabling autonomic applications on the grid. Autonomic Computing Workshop (2003) 48–57
5. Appleby, K. et. al.: Oceano: SLA based management of a computing utility. Proceedings of the IEEE/IFIP International Symposium on Integrated Network Management Proceedings (2001) 855–868
6. Andrzejak, A., Graupner, S., Kotov, V., Trinks, H.: Adaptive Control Overlay for Service Management. Workshop on the Design of Self-Managing Systems, International Conference on Dependable Systems and Networks (DSN) (2003)
7. Leon-Garcia, A., Widjaja, I.: Communication Networks. Mc Graw Hill (2004)
8. Bridgewater, D.: Standardize messages with the Common Base Event model. IBM DeveloperWorks (2004)
9. Talia, D.: The Open Grid Services Architecture: where the grid meets the Web. IEEE Internet Computing Magazine (2002) 67–71
10. Leon-Garcia, A. L. Mason, L.: Virtual Network Resource Management for Next-Generation Networks. IEEE Communications Magazine (2003) 102–109
11. Kreger, H.: Web Services Conceptual Architecture. White Paper, IBM Software Group (2001)