Blueprint for an Autonomic Service Architecture

Ramy Farha and Alberto Leon-Garcia
Department of Electrical and Computer Engineering
University of Toronto, Toronto, Canada
{ramy.farha,alberto.leongarcia}@utoronto.ca

Abstract

Next-generation service providers will offer an array of services and applications that is media-rich, personalized, and context-aware. In this future environment, new services and applications will be introduced, provisioned, operated, maintained, and retired at a pace that tracks changing customer requirements and demands. In order to be cost-effective, these services and applications need to be delivered over an all-IP infrastructure, which is autonomically managed to ensure their delivery at a satisfactory level for subscribed customers. In this paper, we propose a blueprint for a generic Autonomic Service Architecture (ASA) to address these challenges.

1. Introduction

Traditional Service Providers (SPs) are in the midst of a transition from circuit-switched networks, to IP-based packet networks. This transition allows the introduction of a broad range of media-rich, context-aware, and personalized services. However, with the increasing demands for Quality of Service (QoS) by customers, SPs are faced with the challenge of provisioning and managing their service delivery infrastructure in an efficient, cost-effective, and flexible way. Currently, such techniques are mainly manual, leading to slow response times, customer dissatisfaction, and increasing costs. With the changing landscape presaging further complexity of service capabilities, the problem is further compounded. Therefore, SPs need solutions to dynamically marshal their service delivery infrastructure to support the required service mix at a given point in time. In this paper, we propose an Autonomic Service Architecture (ASA), which aims to reduce costs incurred by SPs for service delivery, to improve the customers' experience by guaranteeing pre-defined QoS levels, and to optimize the use of resources at the SP's disposal.

Service delivery follows a lifecycle [1], mainly consisting of the following phases: Creation, Activation, Provisioning, Management, and Termination. ASA proposes a generic architecture to deal with service delivery autonomically. For ASA to operate correctly, SPs will first have to (mostly manually) setup their infrastructure according to the guidelines governing ASA's operation. Then, they can offer services to customers, and use ASA to allow autonomic activation, provisioning, management and termination, without SP's intervention.

Several approaches have recently emerged that relate to ASA, but none is generic enough so it can be applicable to all services, with only minor changes in the setup. The first work in the new wave of autonomics has been the Autonomic Computing proposal by IBM [2]. 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, Autonomic Communication [3], has similar aims to IBM's Autonomic Computing proposal, 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. Furthermore, we consider the interplay of both networking and computing resources to offer services.

In addition, projects such as Autonomia [4], AutoMate [5], and Oceano [6] 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 [7] that are context-aware, and capable of self-configuring, self-composing, and self-optimizing. Oceano is developing a prototype for a scalable infrastructure to enable multi-enterprise hosting on a virtualized collection of hardware resources with dynamic adaptability. HP [8] proposes a service-oriented control system that constantly re-evaluates system conditions and re-adjusts service placements and capacities, organized as an overlay topology with monitoring and actuation interfaces to underlying services. Work involved with policy-based provisioning of computer systems was presented in [9], and in [10]. In the former, policies in a shared computing infrastructure are used to ensure customers receive services with pre-defined levels. In the latter, BPEL workflows [11] are used to provision application services and to automate changes made to this initial provisioning. The difference between ASA and all these approaches is that they consider particular services to which their design is appropriate. In addition, each approach tackles only one phase of the service delivery process.

The main contribution of this paper is that it proposes an architecture to ensure automated delivery of services involving both computing and networking resources. To our knowledge this is one of the earliest works in that direction. In addition, ASA is concerned with several phases of the service delivery process, to present an integral approach to SPs wishing to deliver services over their infrastructure. Our previous work [12] on ASA introduced a high-level overview of the architecture. In this paper, we explain ASA's concepts and design in more details. Due to space limitations, this paper will mainly focus on the architecture. In future publications, we hope to further detail ASA's operation and the algorithms it uses.

2 Autonomic Service Architecture Concepts

In this section, we present the major concepts behind ASA's design, namely considering that "everything is a service", allowing autonomic service delivery, using virtualization, and proposing Autonomic Resource Brokers (ARBs) to regulate ASA's operation.

2.1 Everything is a service

ASA is mainly driven by the view that, for a SP, "everything is a service". Some services can be offered to customers, such as Voice over IP (VoIP), while others are used as components to build other services, such as IP packet transport. This approach is not new, and dates back to telephony services and the Intelligent Network [13] platform. More recently, in the IT world, service oriented architectures have emerged to deliver IT services [14]. ASA is built around the same concept: A service delivered to a customer by a SP is usually formed from a composition of component services. Some services are "atomic", i.e. cannot be broken down into component services anymore, and usually act on the underlying resources. We refer to them as basic services. Other services, composed of several components, are referred to as composite services. Every service consists of an allocation of resource amounts to perform a function.

2.2 Autonomic Service Delivery

Automating the delivery of services involves orchestrating the SP's computing and networking resources to deliver services according to pre-defined service level agreements (SLAs). This goal is achieved through a control loop which monitors the underlying resources, analyzes the situation, plans future actions, and executes them [2]. This autonomic control loop needs to provide autonomic systems with the self-management characteristics: self-configuring, self-optimizing, self-healing, and self-protecting. In ASA, the building blocks of this autonomic control loop are kept as generic as possible to be easily adjustable for all services.

In a real-world environment, it is highly unlikely that a single SP is capable of offering End-to-End (E2E) service delivery, since multiple SPs would usually be involved in the E2E delivery path. Recent work has addressed the issue, using Infranets [15] to provide E2E QoS guarantees. ASA is designed to simplify this E2E service delivery.

2.3 Virtualization

Virtualization is a commonly used approach to decouple the user-perceived behavior of hardware and software resources from their actual implementations. With the increasing number of heterogeneous resources involved in the delivery of a service, ASA uses virtualization to abstract underlying resources (such as servers, routers). A virtualization layer makes this abstraction possible. In ASA, underlying resources appear as a substrate to which requests are sent, and from which responses are received.

All services in ASA could be seen as "Virtual Services", a concept we previously introduced [16], based on the iterative composition of services, as well as an iterative approach for the provisioning, operation, management, and termination of these services. Fig. 1 shows ASA's layered view (top part), as well as the notion of Virtual Services and their layered management (lower part). In the rest of this paper, we refer to Virtual Services as services for simplicity. In ASA, composite services are a QoS-based composition of basic services, hence ASA can be seen as a service oriented architecture (SOA) for telecommunication services.

Physical resources refer to hardware (router, server, switch, link), or software (content) resources at the SP's disposal. Each physical resource can be virtualized into several Virtual Resources (VRs), which can be seen as different capabilities of that same resource. Virtualization facilitates ASA's operation, because it abstracts the underlying resources, and simplifies interaction with them. Regardless of the underlying resources' topology, the virtual resources layer exhibits a single adapter to ASA, as explained in [12]. The interaction between ASA and this adapter follows well-defined formats detailed next, while the interaction between

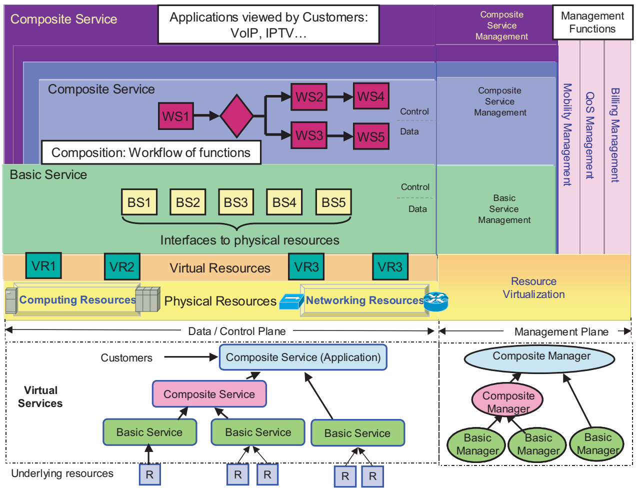

Figure 1. Autonomic Service Architecture

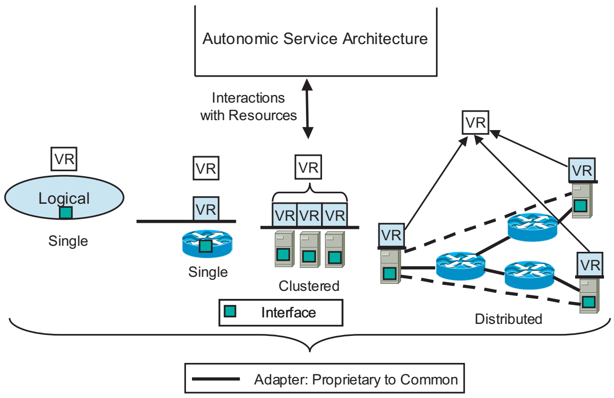

Figure 2. Resource Virtualization

the adapter and the underlying resources depends on the topology at hand. For instance, in Grids, ASA interfaces to the Grid Resource Allocation Manager (GRAM) [7]. Figure 2 shows the virtualization procedure for different underlying resources' topologies.

We define the Common Resource Format (CRF), an XML-based representation, to represent the different types of virtual resources needed for service delivery. CRF needs a mapping function between the absolute metrics and the CRF metrics. We propose to benchmark absolute metrics (CPU speed, RAM) against well-known service needs, in order to obtain service-oriented CRF metrics. For instance, for a VoIP service, it is more useful to know that a server can handle Y voice streams using G.723 codecs.

As well, we define the Common Command Format (CCF), an XML-based representation, to provide a common representation of the actions that need to be performed on the underlying resources. Some commands have arguments, as well as a mapping function between CCF, and the "backend" commands performed on the underlying resources. For instance, for a VoIP service, commands such as "Set" can be destined to an edge router, and the argument could be a DiffServ class, and the criteria for an incoming IP packet to be considered in that class by the edge router.

2.4 Autonomic Resource Broker

As mentioned previously, the main task of ASA consists of automating the delivery of services offered by a SP to customers. This goal is achieved in ASA through a self-managing entity, called the Autonomic Resource Broker (ARB), whose role is to ensure automated delivery of the services by SPs. ARBs handle provisioning, management, and termination of services autonomically, by interacting with underlying resources and component services. The format of the messages sent and received by ARBs are based on the aforementioned CRF and CCF representations for resource quantification and configuration respectively, hence are independent of the actual resources involved in the service delivery. This allows the SP to have an ARB "template", which we detail in the next section, that only needs to be slightly modified for each specific service.

When customers activate service instances they have bought from SPs, these service instances are managed at the SPs by Service Instance ARBs (SIARBs), with a pre-defined aggregation level. This aggregation level corresponds to the grouping of individual service instances (ISIs). The multiple service instances of a particular service offered by a SP (managed by SIARBs) are managed by Composite ARBs (CARBs). In addition, some CARBs, called basic CARBs, manage the basic services which sometimes only consist of virtual resources. Other CARBs, called composite CARBs, manage composite services. The different services offered by a SP (managed by CARBs) are managed by a Global ARB (GARB), which handles all the resources available at this SP's disposal. A layered structure of ARBs is proposed, as shown in Fig. 3.

3 Autonomic Service Delivery using ASA

In this section, we describe ASA, and show how it ensures automated delivery of services. The scenario we consider is that of a SP offering a range of services with different Classes of Service, advertised to customers. CARBs are built per-"Class of Service", and per-"Customer Type". In ASA, SPs specify two main documents to regulate a CARB's operation: SLA Templates and Service Templates, which are stored in the SP's GARB, with local copies available at the appropriate CARBs. As mentioned previously,

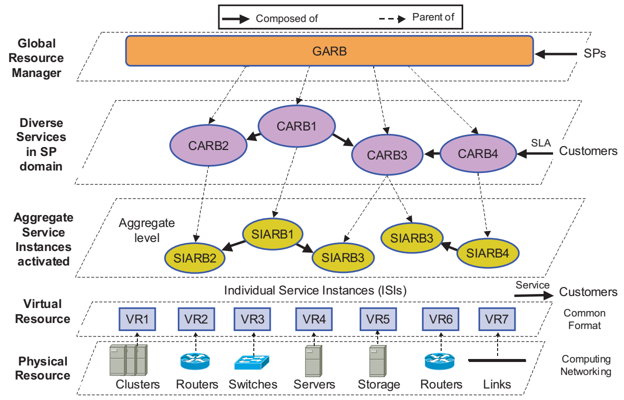

Figure 3. Autonomic Resource Brokers

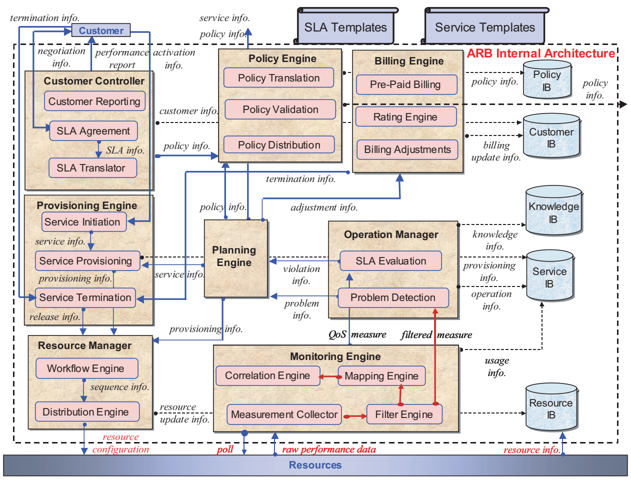

Figure 4. ARB Internal Architecture

service delivery is ensured by the interaction of ARBs forming ASA. The ARB architecture is shown in Fig. 4.

3.1 Templates and Information Bases for Automated Delivery

3.1.1 SLA Templates

Defines SLA Templates for the services offered over ASA. ASA adopts the TMF's classification of customer types [1]: Individuals, Enterprises, and Other SPs. We believe that the expertise level for the three customer types are different, hence it is possible to define three SLA Templates per service offered to customers based on these types.

3.1.2 Service Templates

Defines Service Templates for the services offered over ASA, which are needed to allow ASA to operate autonomically and to ensure delivery of services. Some fields in the Service Templates are fixed, while others are set at runtime, as a result of the SLA negotiated between customers and SPs. Service Templates will provide guidelines for the operation of ASA, and since several Classes of Service are defined for each service offered, a different Service Template is needed for each such Class of Service.

3.1.3 Information Bases

Information Bases are needed to store the information needed by ARBs. Information Bases (IBs) can be classified several logical groupings. Five major information bases are needed: Policy IB (PIB), Customer IB (CIB), Service IB (SIB), Resource IB (RIB), and Knowledge IB (KIB).

3.2 Policy Control

Several policies are created initially by SPs manually, or at runtime as a result of customers buying services. Existing policies can be updated as a result of service demand and load variations. The role of this part is to ensure policies are valid, written in the appropriate format, and distributed to the appropriate component when needed.

3.2.1 Policy Translation

All policies are stored in the PIB according to a pre-defined format [17]. In order to be easily used in the ARB's operation, the role of this part is to check new policies and ensure they are in the appropriate format prior to being validated, and distributed or stored in the PIB.

3.2.2 Policy Validation

Several policies exist in the PIB at any time, and the creation or update of policies could lead to conflicts, redundancy, inconsistency, and infeasibility. The role of this part is to ensure no such problems occur and to remedy to them, using appropriate algorithms [18], prior to storing the validated policies in PIB.

3.2.3 Policy Distribution

When needed, the appropriate policies are distributed to the ARB components or to other ARBs, becoming sentinels that ensure valid ASA operation.

3.3 Customer Control

Customer Control is involved in the interaction between SPs and customers. It consists of the Customer Reporting,

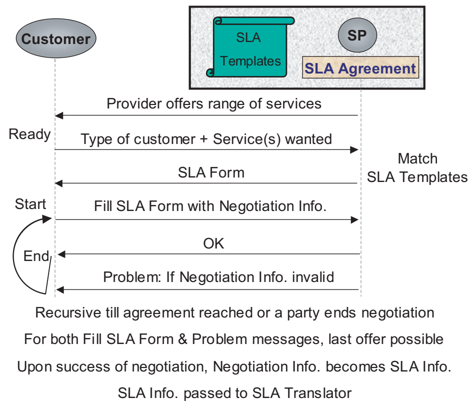

Figure 5. SLA Negotiation Procedure

SLA Agreement, and SLA Translator components. It is important to differentiate between the SLA Agreement component of Customer Control and the Service Initiation component of Provisioning Engine. SLA Agreement is a one-time operation that occurs prior to using the service by the customer. The Service Initiation, on the other hand, occurs each time the customer activates a service instance to use the service he had previously subscribed to.

3.3.1 SLA Agreement

To reach an agreement over the SLA, a simple negotiation procedure is needed. Note that the format and the semantics of the SLA are transparent to the negotiation procedure. In ASA, we propose a simple SLA Negotiation procedure, shown in Fig. 5. SPs keep different SLA Templates for each service offered, depending on the customer types. These SLA Templates will be used to create SLA forms that are filled by customers.

We now briefly explain the operation of the SLA Negotiation procedure. Initially, the SP is offering a range of services to its customers through an interface with "implicit" Classes of Service. At this point, the customer is in the "Ready" state. In the first phase of the negotiation, the customer chooses the service (with the Class of Service) that it would like to subscribe to, sends the request <Customer Type, Service Wanted, CoS> to the SLA Agreement component. The request is matched to the appropriate SLA Template, and an SLA form with empty fields to fill is sent back to the customer, who moves to the "Start" state. In the second phase of the negotiation, the customer fills in this SLA form with the missing information <Customer Personal Information, Service Schedule, Customer Reporting Information,...>. This information, along with the current number of negotiation attempts (referred to as negotiation info.), is sent back to the SLA Agreement component, which verifies that the negotiation info. is valid, based on pre-defined algorithms specified by SPs. These algorithms could be as simple as verifying completeness of the Customer Personal Information, the possibility to offer the service as per the Service Schedule specified, and to ensure that the frequency of Customer Reporting Information does not exceed an overhead threshold. If the negotiation info. is valid and does not violate any policies, the SP informs the customer that the negotiation is successfully completed and that an agreement has been reached. The SP creates a unique identifier for the customer. At this point, the customer is in the "End" state. The negotiation info. validated, along with other fields, now referred to as SLA info., are sent to the SLA Translator component. If the negotiation info. is invalid , the SP refuses the agreement, and sends a Problem notification to the customer, informing him of the fields which cause problems. Alternative values, or bounds on values allowed, could also be proposed. In general, the Problem message consists of: <Problem Identifier, Time that the problem was detected, Gravity of the situation (Relates to attempts), Problematic Fields, Suggestions on values for the problematic fields (Optional)>. This procedure is recursive, and continues until an agreement is reached. However, several ways for SPs to reduce the length of the negotiation are possible. For instance, a limit N on the number of attempts can be used, and when $(N-1)$ attempts are made so far, the SP can then issue a "Last Offer" reply, indicated by the "Gravity of the situation" field in the Problem message. The customer responds either positively (with valid entries in the SLA form), or negatively. In addition, this procedure allows the customer to interrupt the negotiation at any point by sending a "Reject" message as a response to the SP's Problem message. Note also that both the SP and the customer can interrupt the negotiation at any time in the negotiation procedure, by sending a "Stop" message to the other party.

3.3.2 SLA Translator

Once the negotiation procedure between the customer and the SP is completed, SLA info. is sent by SLA Agreement to SLA Translator. SLA info. is parsed in the SLA Translator, and based on the fields parsed as well as the predefined policies regulating the SLA Translator, new policies are generated (if needed) and sent as Policy Info. to the Policy Control for validation prior to being stored in the PIB. The SLA Translator sends the SLA info. to the CIB to store the information based on the Customer Identifier field in SLA info., which is unique. In addition, the SLA Translator fills the updated bill for the current cycle for the particular customer with the flat charge in SLA info. The Billing Engine, based on the billing policies, will generate the billing update info. at a frequency specified in SLA info.

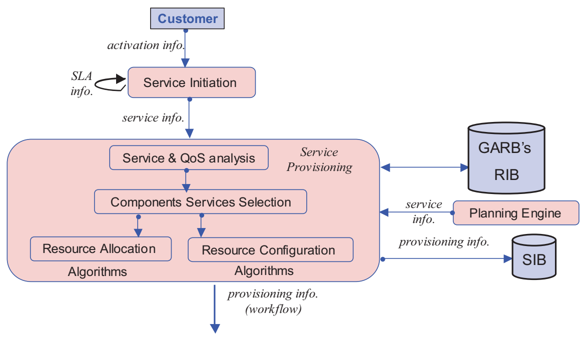

Figure 6. Provisioning Engine Details

3.3.3 Customer Reporting

Customers wish to have access to monitoring results, in order to allow them to switch SPs if performance is not satisfactory. Customer Reporting sends the performance results back to the customers, at frequencies / circumstances resulting from the customer's SLA Agreement with the SP. This performance info. is sent by the SIB when polled by Customer Reporting based on policies created at runtime when SLAs were agreed upon between customers and SPs. When Customer Reporting Information conditions / circumstances are met, the performance info. is polled from SIB based on the unique Customer Identifier, and put in the appropriate format to be displayed to customers.

3.4 Provisioning Engine

Customers activate services bought by contacting the Service Initiation component in the CARB handling the desired service, as shown in Fig. 6.

3.4.1 Service Initiation

The activation info. received contains: <Customer Identifier (Unique), Service Identifier (Unique)>. The initiation request sent by the customer goes through a Call Admission Control (CAC) algorithm, which accepts or not this initiation request. This allows unique matching in the CIB contents, to retrieve the SLA info. In addition, the Service Template contains information that can be retrieved for the service. The SLA info. and the Service Template lead to the service info. sent to the Service Provisioning component for appropriate service provisioning.

3.4.2 Service Provisioning

ASA is built based upon the premise that "everything is a service". Using this approach, composite services are composed out of component services, and provisioning composite services consists of choosing the appropriate virtual resources amounts to allocate to each service component, and the basic services which perform the actions needed to "program" the resources in the order needed.

The Service and QoS analysis component parses service info., to determine the type of component services needed, the order in which these services need to be executed, their QoS requirements, and the constraints on them. For E2E service delivery, the Service and QoS analysis component uses the mapping information in service info. to map the QoS characteristics of individual service components to the E2E delivery QoS requirements. For instance, the compounded E2E bandwidth that can be guaranteed is the minimum of that provided by each component service referring to IP transport that is involved in this E2E path, while the compounded delay that can be guaranteed is the sum of that incurred in each component service referring to any resource which introduced delay in this E2E path.

The Components Services Selection component optimizes the selection process of basic services from all the services at the disposal of the SP, some of which are owned, others were bought from other SPs.

The Resource Allocation component decides on the amount of virtual resources of each type needed for the service instance, based on algorithms providing optimal allocation amounts [19]. These amounts are expressed using CRF. For instance, amount X of VR1, and amount Y of VR2 are needed for this service instance. These values are stored in the ARB's SIB based on the service instance identifier, and the ARB's RIB is updated to reflect the allocation amounts decided by the Service Provisioning component, so that available resource quantities are up-to-date.

The Resource Configuration component decides on the actions to be taken for the provisioning of the service instance, based on algorithms providing optimal scheduling of actions [20], and expressed using CCF. For instance, "Select" server X for this service instance. Some of the possible configuration commands in CCF involve physical resources configuration, changes to call admission control mechanisms at a physical resource, redirection of traffic entering a physical resource, using a particular server / gateway to achieve load balancing. These commands are stored in the ARB's SIB based on the service instance identifier.

The Service Provisioning components use the service info. resulting from the Service Initiation component, and make their decisions based on policies stored in PIB, as well as the status of the resources at the disposal of the SP for this service, stored in the CARB's RIB. The output of this component, called provisioning info., contains information sent to the Resource Manager component for provisioning the underlying resources.

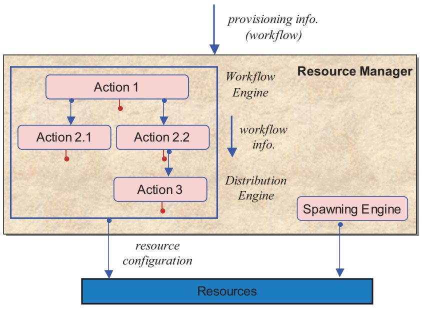

Figure 7. Resource Manager Details

3.4.3 Service Termination

When customers want to stop using a service, or when SPs want to retire a service, they send termination info. to the Service Termination component. The termination info. consists of the Service Instance Identifier, which is unique in the ARB's SIB. The Service Termination component retrieves the provisioning info. from the SIB, parses, modifies, and transforms it into the reverse of the actions taken when the service instance was activated and the resources were allocated / configured. For instance, "Allocate" has "Release" as a converse. The output of this component is release info., which has a similar structure to provisioning info., except that it performs reverse actions. The values in the ARB's RIB for resource amounts are updated by adding the amounts of released resources to the updated resource quantities available.

3.5 Resource Manager

The resource manager is the component communicating with the underlying ARBs/physical resources needed, and taking the actions needed to provision the resources needed by the service instance, as shown in Fig. 7.

3.5.1 Workflow Engine

Given that the Service Provisioning decided on the amounts of resources needed as well as on the actions to be performed on the underlying ARBs / resources, the Workflow Engine supervises the execution of this workflow by sending the actions needed to the Distribution Engine.

3.5.2 Distribution Engine

The actions are distributed to the appropriate underlying ARBs / resources, according to the sequence dictated and controlled by the Worklfow Engine. These actions are sent according to the aforementioned CRF and CCF formats, and an adapter is needed to translate from CCF / CRF to the proprietary formats needed to interface with the physical resources directly (SNMP, CLI, TL1 . . .).

3.5.3 Spawning Engine

The ARBs needed to manage the service instances activated are spawned by the Spawning Engine, following the concept presented for Virtual Networks (VNs) [21]. Due to space limitations, we will omit details of the spawning process, which is inspired by the initial VN design. Spawning provides the child ARB with the information it needs to start delivering the service instance it handles.

After the autonomic provisioning is completed, the underlying ARBs will monitor the operation of the provisioned service instances. Problem are handled at the current ARB level, unless no solution is found, then the parent ARBs will attempt to remedy the situation.

3.6 Monitoring Engine

The Monitoring Engine operation is regulated by policies retrieved from the current ARB's PIB, based on the matching criteria. A monitoring policy indicates: <Measurements Needed, Frequency of polls (per unit time), Need for Notifications (Boolean value)>.

3.6.1 Measurement Collector

Resource measurements are collected by the Measurement Collector component, which is the interface to the underlying measurement infrastructure and receives raw performance data, which it passes to the Filter Engine. Based on the frequency of polls defined by the monitoring policies, the Measurement Collector polls the underlying resources to obtain the required measurements.

3.6.2 Filter Engine

Resource measurements are filtered by the Filter Engine component, and sent to both the Metric Mapping component in Monitoring Engine and to the Operation Manager component. Measurements can be obtained either through polling, notification, or periodically. The algorithms to filter unwanted data are specified. At the ARB components, the tradeoff is between precision and overhead of measurements. The more precise the results are needed to be, the more measurements we need to perform. The filtering procedure is a feedback loop, where the frequency of the filtering measurements and the filtering criteria can be changed based on changing conditions, or after catastrophic events, in order to ensure that the operation is back to normal by changing monitoring policies in the PIB.

3.6.3 Mapping Engine

ASA views resources as virtual, consisting of an abstraction of physical resources. The measurement infrastructure collects raw performance data. Some are network-level metrics, such as CPU utilization, bandwidth, queue lengths, memory utilization. Others are application level metrics such as application response time, number of sessions, transaction rate. Hence, there is a need to map network and application measurements (raw measurements) to performance metrics (QoS parameters), as illustrated in the Service Template, for the Operation Manager component to analyze them. QoS parameters can be obtained by mappings ranging from simple operations such as summation, average, maximum, or minimum of a group of measurements collected, to more elaborate mapping algorithms [22].

3.6.4 Correlation Engine

Prior to sending the performance metrics to the Operation Manager component, we detect some situations that might help reduce the overhead and the information exchanged in the ARB. For instance, some measurements may be redundant, some may be received late and may obsolete previous measurements on the same resource, which are not obvious to detect. Algorithms are defined with different intelligence levels to handle this problem using techniques such as spatial or temporal correlation [23]. The QoS Measure sent to the Operation Manager contains measured QoS parameters related to the service managed by the current ARB.

3.7 Operation Manager

3.7.1 Problem Detection

Several problems occur in the SP domain. Faults can occur when computing or networking components fail. Overloads can occur when the demand on a particular component (computing or networking) exceeds the capacity of the component. Congestion can occur when the performance of some components (computing or networking) degrades because of excessive load. For instance, the Problem Detection could be as simple as comparing load measures for underlying resources vs. thresholds allowed by policies.

The result of the operation performed by this component is sent to the Planning Engine as problem info., containing the Virtual Resource Identifier, along with the problem type (Fault / Overload / Congestion) detected.

3.7.2 SLA Evaluation

Based on the QoS measure obtained from the Monitoring Engine, SLAs are evaluated following mapping of QoS parameters to the SLA for that service instance. An SLA violation detection test is performed by comparing this evaluated SLA with the agreed upon SLA between customers and SPs. Violations that are detected are sent to the Planning Engine for appropriate plan elaboration, as a violation info. message. The violation info. needs to indicate the time of occurrence of the violation, and the SLA parameters violated.

Note that the operation manager stores information on occurring events in the KIB. This information will become useful in future decisions, as it can provide guidance in case similar occurrences are noticed. This knowledge info. stores the Service Instance Identifier in case the information is needed in the current service instance lifecycle, the Service Identifier in case the information is needed after the current service instance lifecycle is completed, the knowledge information to provide a unique match in order to recognize the conditions under which previous solutions were applied, and the time at which the entry was generated.

3.8 Planning Engine

Fig. 8 shows ARB's Planning Engine. The inputs to the Planning Engine are:

-Customer entry, i.e. customer information related to services where problems have occurred, and the SLA corresponding to these services, obtained from the CIB.

-Service entry, i.e. performance requirements for the service (SLAs), obtained from the SIB.

-Policy entry that constrains solutions, i.e. policies that restrict resource allocation, obtained from the PIB.

-Resource entry extracted from the existing resource pool keeping track of available virtual resources at the SP's disposal obtained from the RIB.

-Knowledge entry containing previous comparable situations, where the advocated solutions could be used instead of elaborating new ones, obtained from the KIB.

-Problem info., and Violation info. from the Operation Manager component to indicate problems detected, and SLA violations.

If the problem cannot be handled at the current ARB level, the Planning Engine in the parent ARB is notified. As well, the child ARBs might need to be modified so the Planning Engine notifies the appropriate components in these child ARBs. If the Problem info. and Violation info. sent by the Operation Manager component could be matched to a previous occurrence from the KIB, the solution is retrieved and sent to the Service Provisioning component for re-provisioning.

Here is a list of the components that the Planning Engine consists of, as well as their output:

-A Policy Update Engine that decides whether changes to the policies regulating the ARB components is needed,

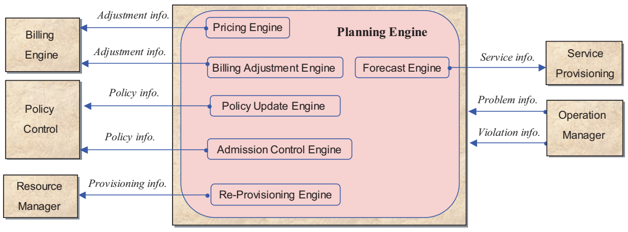

Figure 8. Planning Engine Details

and sends suggested changes to the Policy Control component in the form of policy info. For instance, changes to the Monitoring Engine, such as frequency of measurements and type of resources measured for instance asking for additional monitoring data by increasing the frequency of raw performance data collection.

-A Billing Update Engine which sends the adjustments needed as Adjustment info. to the Billing Engine component which updates the billing records. These adjustments are mainly based on the violations detected by the Operation Manager. In addition, these adjustments could also include changes to the Billing Engine's operation.

-A Re-Provisioning Engine which periodically analyzes the performance. Even when no problems or violations are signaled by the Operation Manager, the Planning Engine ensures resources are optimally allocated by calculating the optimal resource allocations. The Performance Evaluation sub-component in the Re-Provisioning Engine is not triggered by any other component in ASA. Instead, it periodically runs an optimization algorithm based on the current status of the resources available (obtained from the RIB), and the current resource allocation and configuration (obtained from the SIB). If the current resource distribution is not optimal, a new allocation is found using this optimization algorithm. The result from this component needs to be input to the Resource Manager as Provisioning info. to reallocate the resources accordingly.

-A Forecast Engine, which, based on the inputs, generates a "Traffic Matrix" type of output. This output is in fact a demand for new services, which are either owned by the SP owning the ARB that generated the "Traffic Matrix" demand or not. In both cases, we follow a Customer-SP type of approach and request this new service, through the "Service Provisioning" component, in other words, generating a new Service Info.

-A Pricing Engine which dynamically determines the price of resources offered by ASA based on pricing policies and algorithms. We relate the Pricing Engine to the Call Admission Control (CAC) in the Service Initiation component, to allow a dynamic and adaptive pricing as a natural means for admission control.

-An Admission Control Engine which indirectly modifies the operation of the Call Admission Control (CAC) in the Service Initiation component, by changing the policies related to it in the Policy Control component.

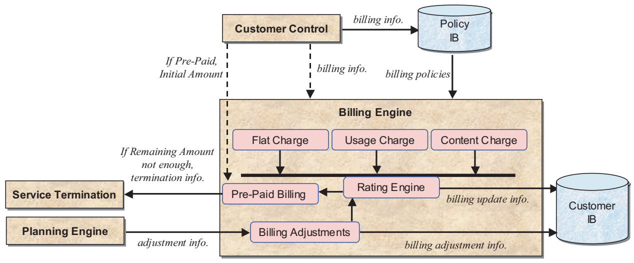

Figure 9. Billing Engine Details

3.9 Billing Engine

Fig. 9 shows ARB's Billing Engine. The charge to the customer consists of three parts: A flat charge, a usage charge, and a content charge. A usage analyzer and a content analyzer in the Monitoring Engine are used to allow flexibility in using the appropriate billing solution. The Billing Engine contains a Pre-paid Billing component to allow an additional type of billing plan. When customers negotiate the SLA with the Customer Control, billing policies are generated that regulate how the particular customer is going to be billed. The exact details of the "billing plan" will be determined when the customer initially signs up for the service, and the contract contains fields which generate billing policies on the fly.

4 Conclusion

The road towards fully autonomic service architectures is still long. However, we hope that in this paper, we were able to present an autonomic service architecture (ASA) to automate the activation, provisioning, management, and termination of services. ASA is based around concepts of service oriented architectures, virtualization, and E2E service delivery. As current and future work, we are studying the CRF and CCF representations for virtualizing underlying resources, and designing elaborate algorithms to provide the degree of self-management needed in every autonomic system. We are also examining possible approaches to resource allocation, correlation of monitoring measurements, problem detection, planning, as well as completing a prototype implementation of ASA in the Network Architecture Labs at the University of Toronto, based on open standards such as Web Services.

References

[1] TeleManagement Forum, "SLA Management Handbook: Version 2.0," Apr. 2004.

[2] IBM Corporation, "An architectural blueprint for autonomic computing." White Paper, Apr. 2003.

[3] A. C. Forum, "Autonomic communication." http://www.autonomic-communication.org.

[4] D. Xiangdong et. al., "Autonomia: an autonomic computing environment," in *Proceedings of the IEEE International Performance, Computing, and Communications Conference*, pp. 61–68, Apr. 2003.

[5] M. Agarwal et. al., "AutoMate: enabling autonomic applications on the grid," in *Autonomic Computing Workshop*, pp. 48–57, June 2003.

[6] K. Appleby et. al., "Oceano: SLA based management of a computing utility," in *Proceedings of the IEEE/IFIP International Symposium on Integrated Network Management Proceedings*, pp. 855–868, May 2001.

[7] D. Talia, "The Open Grid Services Architecture: where the grid meets the Web," *IEEE Internet Computing Magazine*, vol. 6, pp. 67–71, Nov. 2002.

[8] A. Andrzejak, S. Graupner, V. Kotov, and H. Trinks, "Adaptive Control Overlay for Service Management," in *Workshop on the Design of Self-Managing Systems, International Conference on Dependable Systems and Networks (DSN)*, June 2003.

[9] K. Appleby, S. B. Calo, J. R. Giles, and K. Lee, "Policy-based automated provisioning," *IBM Systems Journal*, vol. 43, no. 1, pp. 121–135, 2004.

[10] A. Keller and R. Badonnel, "Automating the Provisioning of Application Services with the BPEL4WS Workflow Language," in *Proceedings of the 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, Nov. 2004.

[11] IBM Corporation, "Business Process Execution Language for Web Services." www.ibm.com/developerworks/library/ws-bpel, 2004.

[12] R. Farha, M. S. Kim, A. Leon-Garcia, and J. W. K. Hong, "Towards an Autonomic Service Architecture," in *Lecture Notes in Computer Science, Volume 3751*, pp. 58–67, Oct. 2005.

[13] R. B. Robrock, "The intelligent network-changing the face of telecommunications," in *Proceedings of the IEEE*, vol. 79, pp. 7–20, Jan. 1991.

[14] IBM Corporation, "Service Oriented Architecture." http://www-306.ibm.com/software/info/openenvironment/soa/.

[15] Juniper Networks, "Defining Infranets: Evolution and Milestones." White Paper, Aug. 2004.

[16] R. Farha, M. S. Kim, A. Leon-Garcia, and J. W. K. Hong, "Autonomic Service Architecture using Virtual Services." Proceedings of Asia & Pacific Network Operations & Management (APNOMS), 2005.

[17] D. Agrawal, K. Lee, and J. Lobo, "Policy-based Management of Networked Computer Systems," , vol. 43, Oct. 2005.

[18] E. C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," *IEEE Transactions on Software Engineering*, vol. 25, pp. 852–869, Nov. 1999.

[19] W. E. Walsh et. al., "Utility functions in autonomic systems," in *First International Conference on Autonomic Computing*, pp. 70–77, 2004.

[20] J. Yu, R. Buyya, and C. Tham, "QoS-based Scheduling of Workflow Applications on Service Grids." Technical Report, GRIDS-TR-2005-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, June 2005.

[21] A. Leon-Garcia and L. Mason, "Virtual Network Resource Management for Next-Generation Networks," *IEEE Communications Magazine*, vol. 41, pp. 102–109, July 2003.

[22] H.-J. Lee, M.-S. Kim, J. W. Hong, and G.-H. Lee, "QoS Parameters to Network Performance Metrics Mapping for SLA Monitoring," *KNOM Review*, vol. 5, pp. 42–53, Dec. 2002.

[23] K. Appleby, G. Goldszmidt, and M. Steinder, "Yemanja – a layered event correlation engine for multi-domain server farms." Integrated Network Management VII, IEEE, 2001.