Network Working Group
Internet Draft
Intended status: Standards Track
Expires: July 2012

A. Dalela
Cisco Systems
M. Hammer

January 4, 2012

SOP Network Architecture
draft-dalela-sop-architecture-00.txt

Abstract

Cloud services need to interoperate across cloud providers, service vendors and private/public domains. To enable this interoperability, there is need for network level deployment architecture to connect users and providers. This document describes functionality partitioning in network deployment and the different advantages of using distributed functionality deployments.

Table of Contents

1. Introduction

This document describes network architecture and deployment models for supporting service orchestration. The architecture and deployment models are driven by the following main requirements:

- Security of the Service Network. A provider needs to ensure that its service network is secure from external attacks. To this end, the provider may need to hide service topologies, and inspect or

modify service transactions on the fly. Network architecture will define how topology hiding and security are achieved.

- Policy Control of Service Network. Service deployments may cross customer and provider boundaries as described in Service Orchestration Protocol (SOP) requirements [REQT]. Each customer or provider may wish to enforce policy rules for service usage at ingress and egress points. An architecture definition will define how customers and providers can policy control services.

- Separate Service-Dependent and Service-Independent functions in the network. A service consumer or provider should not have to upgrade their orchestration infrastructure in order to deploy or use a new service. Separation allows new service deployment without disrupting the network of existing services.

- Scaling the Service Network. To scale service across many consumers, service type and locations, distribution of service functionality is needed. For instance, service-dependant intelligence for a set of customers might be stored in one network element. Or, all intelligence related to one class of services may be centralized in one network element. Or, all customers in one particular geography or location may access services from one particular network element. Service orchestration architecture should support these and other types of deployment models.

- Bundling and Tiering of Services. Services often come in bundles. If we provision a Virtual Machine, we also need to provision storage, QoS, access control, intrusion prevention, etc. Some services may need to use other services in turn. Architecture should define how service tiering and bundling is achieved.

- Service Network Reliability. When services are critical to the working of an organization, it is important to define how users will be able to receive continued service even in case of network failures. Architecture should help system reliability.

This document describes a SOP network architecture that can be employed to fulfill these requirements.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

3. Terms and Acronyms

The key words Provider, Vendor, User, Orchestration, Client, in this document have the same meaning as defined in SOP requirements [REQT].

Service Node (SN): A Service Node is any hardware or software entity that can be made available for use as a service. A Service Node may be associated with an "agent" that receives orchestration requests and may in turn execute them on behalf of the service. Use of an agent abstracts service logic from rest of the network.

Proxy - this is the service-independent network element that accepts Client or other Proxy requests and forwards them to services or other proxies. The Proxy may forward requests to SNs, Workflow Servers or other Proxies. A network of interoperable clouds may use multiple proxies. A Proxy may inspect or modify packets in transit.

Workflow Server (WS) - this is the service-dependent network element and contains Workflow definitions along with policies to validate or modify Workflow requests.

Requesting Proxy (RP) - this is the SOP Proxy that makes service requests. The RP may forward them to the Locating Proxy (LP).

Locating Proxy (LP) - this is the SOP Proxy that performs load-balancing, locating and security functions in SOP network. It forwards received requests to the Serving Proxy (SP).

Serving Proxy (SP) - this is the SOP Proxy that actually controls a particular type of service orchestration.

Requesting WS (RWS) - this is the WS associated with the RP.

Locating WS (LWS) - this is the WS associated with the LP.

Serving WS (SWS) - this is the WS associated with the SP.

Workflow Anchor (WA) - this is the Proxy that controls the Workflow execution. All Task branching must take place at the Anchor.

4. Problem Statement

The key problem for orchestration architecture is how to ease the creation of new services by (a) tiering service one on top of another, (b) bundling multiple independent services as a single use-case, (c) customization of a single service for different users.

Tiering refers to one service using another service for its functioning (e.g. SaaS might use PaaS, PaaS might use IaaS, and SaaS or PaaS will not work without IaaS). Bundling refers to two or more services that work independently but have improved functionality by combining (e.g. network, security and compute combine to deliver improved functionality to the user). Customization means adding functionality to a service without tiering or bundling (e.g. a security device may be improved with intrusion detection).

A realistic service deployment requires complex service combinations. E.g., if we are provisioning a VM, we may also need to provision network attached storage, security rules that limit access to that storage, firewalls and access control that restricts access to the VM, bandwidth from a suitable starting point to the VM, user provisioning for who can access the VM, load-balancers and WAN optimization techniques, intrusion detection and prevention or techniques to log and report accesses to a service, etc.

Currently, services are combined by combining lower level APIs into higher-level APIs. The inputs to these APIs must follow several cross-domain semantic "rules", without which the API combination will not give a usable service. For example, the IP of a host must belong to the subnet on the switch, because otherwise packets will never be routed. The IP access-list on the switch must permit the IP address on the host, otherwise packets will be dropped. Open ports on a host must also be open on the firewall because otherwise packets will never be received. There are many such "rules" to be followed. Each customer may design their own rules, restricting the service design per their needs. APIs don't allow us to define these rules in the abstract. They must be embedded in the input to the APIs.

Orchestration rules deal with the "semantics" of a service, while APIs deal with the syntax. For any given semantics, there are several possible syntaxes to achieve it. Current cloud technology focuses on the syntax and not the semantics. Thus, the same service will require use of different APIs across different providers. This is unnecessary because the users don't care about the APIs. They care about the service semantics, and the rules of orchestration. How these rules are implemented, is totally irrelevant from the user viewpoint.

From a provider's viewpoint, it is important to integrate hardware and software from many possible vendors in the same way. To achieve this, it is important to have a standard interface to that hardware and software that the provider can use to orchestrate. As long as the interface is standard, it does not matter which type of API flavor is used to control the hardware or software. The provider does not care how a hardware or software vendor implements the interface on their side, as long as the standard interface is available.

```
                       +---------------------+
                       |  Service Semantics  |
       Customer View   | Orchestration Rules |
                       +---------------------+
                                  |
                                  |
                       +---------------------+
                       |   Service Control   |
    Implementation View| Software API Format |
                       +---------------------+
                                  |
                                  |
                       +---------------------+
                       | Standard Interface  |
        Vendor View    |  Protocol Messages  |
                       +---------------------+
```

Figure-1: 3-Tier Orchestration Stack

Figure-1 shows an ideal 3-tier orchestration stack, where users provide orchestration rules and vendors provide a standard interface to their services. In the middle of the orchestration stack are software objects that translate the user given rules into standard interfaces to the multi-vendor services. The properties of the software objects are not visible either to the users or to the service vendors. This decoupling between different views allows independent evolution and backwards compatibility. It means that the orchestration rules are portable and a user can take get the same services from different providers. The provider can integrate many software-hardware products. The software that links semantic rules with the standard service interfaces can evolve independently.

In current cloud technology, all three orchestration tiers are collapsed into a single API construct. The user sees APIs which is an implementation view stretching all the way into the service itself. If the API changed, the user will need to adapt to a new API and the vendor will have to implement the new API. This makes cloud extremely

inflexible to interoperation across boundaries. The central problem of cloud control is that we need to move a single-tier orchestration model to the 3-tier model shown in Figure-1.

Use of APIs as end-user interface hinders fast service customization. Each user encodes their service rules using APIs, but the toolset available to the user can only validate the syntax of the API but not its semantics. That means while the APIs can be compiled to be syntactically correct, the user cannot know that on executing those APIs the service will work as desired. We need a "semantic compiler" to achieve that. The rules in the "semantic compiler" will need to validate that the IP on the host is part of the subnet on the switch, that the ports on the host are open on the firewall, that IP of the host is part of permitted access-lists, etc. This semantic compiler can be customized by each provider and customer. That is, they can define their own rules of orchestration.

```
           Customer                             Provider
        +-------------+                    +-------------+
        | Service     |  Brick and Mortar  | Simple      |
        | Combination |--------------------| Services    |
        | Complexity  |   Low-level APIs   | Supported   |
        +-------------+                    +-------------+
```

Figure-2: End-User Deals with Complexity

In the API approach, a provider exposes low-level APIs using which developers can build customized services. This approach pushes the complexity of service creation from the provider into the customer. A customer has to invest in building these custom solutions and thereby be responsible for the design of service. This is shown in Figure-2 where a customer owns the complexity of service creation.

```
           Customer                             Provider
        +-------------+                    +-------------+
        | Customized  |     Customized     | Service     |
        | Service     |--------------------| Combination |
        | Combinations|  High-level APIs   | Complexity  |
        +-------------+                    +-------------+
```

Figure-3: Provider Deals with Complexity

If, however, the provider decides to incorporate these custom solutions as part of their offerings, the complexity in the Orchestrator grows rapidly. The complexity grows linearly with the number of unique combinations. The complexity grows exponentially with the number of services within a combination. The rapid increase

in complexity makes services brittle and hard to modify. This is shown in Figure-3 where the provider owns the complexity of service variety being created for various types of customers.

Both architectures for building new services are not optimal. They either complicate the user-side or the provider-side of services. The 3-tier architecture shown in Figure-1 on the other hand provides an optimal scheme to creation of new services.

5. Solution Approach

Our solution involves the implementation of the 3-tier orchestration architecture through two functional components shown in Figure-4.

```
    +---------------------+            +---------------------+
    |  Service Semantics  |            |   Workflow Server   |
    | Orchestration Rules |            |=====================|
    +---------------------+            |  Service-Dependant  |
               |                       | Orchestration Rules |
               |                       |   User + Provider   |
    +---------------------+            | Service Definitions |
    |   Service Control   |            +---------------------+
    | Software API Format |            +---------------------+
    +---------------------+            | Orchestration Proxy |
               |                       |=====================|
               |                       | Service-Independent |
    +---------------------+            |    Orchestration    |
    | Standard Interface  |            |  Execution Engine   |
    |  Protocol Messages  |            | Protocol Interfaces |
    +---------------------+            +---------------------+
```

Figure-4: Orchestration Components

The service-independent network element (Orchestration Proxy) is unaware of service, user or policy nuances. It is the execution engine for orchestrating services, and has protocol interfaces to other Proxies or to service end-points. The orchestration rules of a service are defined in the service-dependent network element (Workflow Server). This contains all the policies according which the service will be orchestrated. The Workflow Server delivers an orchestration "master-plan" or workflow to the Orchestration Proxy to execute. The Proxy executes the workflow and reports back status.

The service, user and policy information is encoded in the Workflow Server in an abstract language like XML. We term an XML document that describes a service bundle a "Workflow". Workflows are comprised of Tasks, which represent individual acts of orchestration on individual

service elements. A Workflow represents the order in which the Tasks must be executed. Each Task in the Workflow contains a definition of actions that need to be taken for that task. Each Workflow can be referenced by a Workflow Name. A user requests a service bundle by invoking the workflow referenced by the Workflow-Name.

```
       Customer                             Provider
    +--------------+                    +---------------+
    | Workflow     |                    | Workflow      |
    | Server       |                    | Server        |
    +--------------+                    +---------------+
           |                                  |
           |                                  |
    +--------------+                    +---------------+  +-----------+
    | Service      |   User Friendly    | Service       |  | Simple    |
    | Orchestration|--------------------| Orchestration |--| Services  |
    | Proxy        |  High-level APIs   | Proxy         |  | Supported |
    +--------------+                    +---------------+  +-----------+
```

Figure-5: Complexity Abstracted into XML Workflows

One or multiple Workflow Servers (WS) may be situated at various points in the network. The WS can dereference a Workflow Name into an XML document comprised of Tasks. The WS can perform service-specific validations of the Workflow request against the service specific syntax and semantic rules stored in the WS. If the language to describe syntax and semantics of services has been standardized, the WS can support any number of new service combinations through configuration in the WS alone. These configurations will create a new Workflow in the WS, and specify rules for validating it. Neither the Proxy nor the WS need to be upgraded to support new services.

With a XML definition of Workflows and Tasks, validation rules can be easily defined. For example, XML Schema Definition [XSD] can be used to validate syntax and Object Constraint Language [OCL] to validate semantics of a Workflow request. When a Workflow request arrives, the requested Workflow can be validated against the associated schemes already defined in XML or other abstract languages.

If a request fails validation according to syntax and semantic rules already defined in the WS, the request MAY be rejected. The rules may also specify how to modify all or selected requests before forwarding them downstream. The collection of all the syntax and semantic rules constitute the "policy" framework for service orchestration. This policy framework may be centralized at the provider, at the customer or distributed between the provider and customers. Accordingly, the requests may be modified and/or validated multiple times.

To add a new service, we don't need to create a new API combination. Rather we can add (through configuration) new XML documents to define a service and validate syntax and semantics of the service request. New services can thus be added through configuration alone. With the ability to define and validate Workflows, a user or provider can create service bundles on-demand. The WS stores all user or provider defined Workflows, which are referenced by a Workflow-Name.

6. Architecture Description

This section covers two architecture flavors. First, we describe a Functional Architecture that partitions SOP functionality into separate network elements. This partitioning achieves the service-dependant and service-independent separation at the network level. Second, we describe a Deployment Architecture that allows a provider to scale their service network with security and policy control.

6.1. Functional Architecture

```
                    |                       |                       |
       Customer     |       Provider 1      |       Provider 2      |
                    |                       |                       |
      +----------+  |     +----------+      |     +----------+      |
      | Workflow |  |     | Workflow |      |     | Workflow |      |
      |  Server  |  |     |  Server  |      |     |  Server  |      |
      +----------+  |     +----------+      |     +----------+      |
           |        |           |           |          |            |
           |        |           |           |          |            |
       +-------+    |       +-------+       |      +-------+        |
       | Proxy |----|-------| Proxy |-------|------| Proxy |        |
       +-------+    |       +-------+       |      +-------+        |
           |        |       /   |   \       |      /   |   \        |
           |        |      /    |    \      |     /    |    \       |
           |        |     /     |     \     |    /     |     \      |
       +-------+    | +----+ +----+ +----+  | +----+ +----+ +----+  |
       | User  |    | | SN | | SN | | SN |  | | SN | | SN | | SN |  |
       +-------+    | +----+ +----+ +----+  | +----+ +----+ +----+  |
                    |                       |                       |
```

Figure-4: SOP System Architecture

The SOP network comprises of two types of network elements: a SOP Proxy and a Workflow Server (WS). The Proxy deals with service-independent aspects of orchestration while the WS contains the service-dependant features. These two elements interact using SOP.

6.1.1. Proxy

A Proxy performs Service Routing. When users request a service, the Proxy will route the request to the right location. To do service routing, the Proxy discovers and monitors services. It authenticates services and users, so that unwanted users or services cannot get into the service routing database. A Proxy can forward packets to Service Nodes, Workflow Servers, other Proxies, and responses back to Proxies and Users. The Proxy may statefully inspect packets and insert or remove headers. But, the proxy does not have service-specific intelligence to orchestrate services.

Proxies may operate in one of two modes: Transparent and Aggregated. A Transparent Proxy does not aggregate services; it only forwards all messages transparently. An Aggregated Proxy would aggregate services and publish them as aggregates. In the Aggregated Mode, a single Proxy may publish a wide variety of services to their users, although these services are in turn managed by other Proxies.

A Proxy is expected to perform the following functions:

- Advertize its presence as a network element that is capable and willing to Proxy for service transactions for certain types of services (service-specific Proxies should be possible).

- A Proxy should have the ability to connect to other Proxies and exchange service related information in a way similar to how it might do with a service client. This allows a distributed network of service proxies to be built that exchange service information.

- Discover servers and their service capabilities. After discovery, a Proxy should create a Service Registry that clients or other Proxies can query to discover types of services available.

- Discover user locations and presence status through user registrations and indication of service interests. Location is often a key consideration in delivering services, since certain services may be routed differently based on locations or may be forbidden access or allowed restricted access only.

- Ability to connect to policy databases that determine the rules by which a service request is routed to the next hop, or how a service request may be modified or translated prior to routing. Policy routing of service requests is a very key function by which customers and providers can control service transactions through single points of control in the network. Details on the nature of these policy databases should be described separately.

- Authentication, Authorizing and Accounting of services should be performed by the Proxy. A Proxy may intercept requests and challenge the client to authenticate itself. It may then lookup a policy database to validate that the user is authorized to make service transactions. After services have been created, periodic accounting must also be performed. Existing protocols such as RADIUS or DIAMETER should be used for this purpose.

6.1.2. Workflow Server

The WS stores service-dependant intelligence. This includes service authorization, SLAs for those services, location preferences, charging policies, processes for fulfilling requests, etc. These policies should be available as syntax and semantics validation procedures that can be applied by the WS to received requests. If the validation fails, the requests may be rejected.

To apply customer specific rules, the WS must know about users and their location. To apply service specific rules, the WS must know about service availability and location. The WS should receive this information from the Proxy on a periodic basis or whenever it changes. The WS MUST subscribe for updates with the Proxy.

A WS may automatically trim or expand a requested Workflow prior to execution. For example, to a VM creation workflow, the WS may add tasks for storage, access control, QoS, load balancing, etc. even though the end-user hasn't requested them. These policies may be defined by the user, by the provider, or mutually agreed upon as a SLA. This greatly simplifies the creation of service bundles.

Once a Workflow has been defined, the Proxy or Service Nodes can download the Workflow and Task definitions from the WS. In case of failures or service termination, the WS will determine action reversal, by flipping the individual tasks in the correct order.

6.2. Deployment Architecture

Proxies and WSs may be designated for specific customers, providers, service types, etc. One or both of these network elements may be deployed at multiple points in the network, including but not limited to (a) customer/provider egress, (b) customer/provider ingress, (c) customer/provider intranet. The Proxy and WS interaction MAY be any-to-any. Thus, one Proxy may interact with multiple WSs for multiple service types. Similarly, a WS may serve one type of service-dependant rules to multiple Proxies that support the same service.

If proxy does not interact with any WS, it acts as a stateless-proxy that will not modify the orchestration messages in transit.

6.2.1. Types of Proxies

We define three types of Proxies based upon their different roles in a service network. These types are shown in Figure-5 and described in detail in the following sections. Note that these are logical and functional distinctions and one physical network element could play the role of multiple Proxies, or all Proxies may be combined into a single network element in an implementation.

The separation between types of Proxies makes it easier to discuss the functions of a Proxy in different kinds of deployments. Each Proxy may be managing some SNs and/or Users.

```
   +-------------+           +---------------+          +---------------+
   | Requesting  |           |   Locating    |          |    Serving    |
   |  Workflow   |           |   Workflow    |          |   Workflow    |
   |   Server    |           |    Server     |          |    Server     |
   +-------------+           +---------------+          +---------------+
          |                          |                          |
          |                          |                          |
   +-------------+           +---------------+          +---------------+
   | Requesting  |           |   Locating    |          |    Serving    |
   |    Proxy    |-----------|     Proxy     |----------|     Proxy     |
   +-------------+           +---------------+          +---------------+
```

Figure-5: Three Types of Proxies

6.2.1.1. Requesting Proxy (RP) and WS (RWS)

This Proxy is typically located at a Client premise. The Client may be an end-user, a provider who sources services from other providers and combines them with their own services, or a 3rd party provider who only aggregates services from multiple providers. In these cases, the RP would be located at the edge of a customer or provider's networks, and play functions of egress control.

A RP may also be located inside a provider's network, when it requests services from other Proxies. For example, a SaaS Proxy may be a RP to query IaaS services, for its SaaS services. In this case, the Requesting Proxy would be located inside a provider's network.

The RWS associated with the RP should be used to validate the requests before allowing them to be forwarded. For example, the RWS may be used to validate a customer's requests against the policies

provisioned for a given user before forwarding them. Or, the RWS may be used by a provider to validate provider-to-provider requests before sending them to another provider. The validations may be defined unilaterally or based upon agreements or SLAs defined prior.

6.2.1.2. Locating Proxy (LP) and WS (LWS)

The LP aggregates service information and publishes aggregated information outside. The LP may be used to hide the service topology inside a provider. For instance, a provider who supports multiple services, may only publish its LP address and internally route the requests to a dynamically or policy-selected SP. The LP can also be used to implement high-availability of services by routing service requests to where the requests are best fulfilled.

The LP may statefully inspect received requests and authenticate the senders before forwarding the requests further. It may rate-limit requests from a particular source, and may have the intelligence for intrusion detection and prevention, for protecting the internal network from denial of service (DoS and DDoS) attacks, etc. The rest of the network behind the LP may then assume that packets entering them are secure. This obviates need for every service authenticating the user requests, and building security defenses.

The LWS associated with the LP may have rules to forward the requests based on policies. For instance, the LWS may determine that service requests from a certain customer must be always directed to a definite location. The LWS may choose the nearest service creation location, or route to a location that has resources available.

6.2.1.3. Serving Proxy (SP) and WS (SWS)

The SP may be defined to cater to specific customers, service-types or locations. For instance, there may be a SP for a group of customers who have been guaranteed a certain level of service. A SP may orchestrate services in a given geography. Or, it may deal with a specific type of services such as compute or storage.

The SP can control orchestration across multiple resources or resource domains. For instance, a SP may orchestrate services across compute, storage, network, security, etc. It can also delegate the tasks of individual service components to that domain specific orchestrator. It may instantiate multiple service instances based on a single request, and may be used to setup a complete virtual datacenter on a single request. It shall handle error or failure scenarios and trigger rollback actions. The SP must discover services, allow service registrations, and publish aggregates of

services to upstream LPs. The SP is also responsible for accounting for service usage and must generating charging records.

A SP may become the anchor for complex service orchestration that are outside the domain of its own control. In this case, the SP will play the role of a RP, and make requests to a LP to route requests outside its domain of control. For instance, a SP that creates a pool of virtual machines may run out of resources and direct a request to a LP to find it additional resources in another location.

The SWS associated with the SP should have rules to orchestrate particular types of service requests. It would perform per-service validations of Workflows, checking if a user is authorized to make those service requests, and if those requests are well-formed. Service-specific syntax and semantic rules reside in the SWS. It may modify Workflow requests based on service or user policies.

6.2.2. Interconnecting Proxies and WSs

The connections between Proxies and WSs may not be one-to-one as shown above. A Proxy may connect to multiple WSs for different Workflows and a WS may serve multiple Proxies. A distribution of Proxies and WSs creates optimal load-balancing.

Each WS SHOULD advertize the Workflows it supports to the Proxy. The WS SHOULD also advertize class of users and service domains it can support. A Proxy can use this information to forward appropriate service requests to appropriate WSs. The WS and Proxies use SOP to exchange orchestration information.

```
            +--------+        +--------+         +--------+
            |   WS1  |        |   WS2  |         |   WS3  |
            +--------+        +--------+         +--------+
                | \                |  \                |
                |  -------------\  |   -------------\ |
            +--------+        +--------+         +--------+
            | Proxy1 |--------| Proxy2 |---------| Proxy3 |
            +--------+        +--------+         +--------+
```

Figure-6: Many-to-Many Relation between Proxy to WS

A Proxy must also advertise the Workflows it receives from the WS to other Proxies. This way, a Proxy can know how to reach the Proxy that can execute a Workflow, and be able to route requests to it.

6.2.3. Workflow Branching and Anchoring

When a complex Workflow spans across multiple SNs (compute, storage, network, security, software, etc.) there has to be a point where the Workflow is broken into individual Tasks for execution. We call the decomposing of a Workflow into Tasks "Workflow Branching". The point in the network where the Workflow is decomposed into Tasks is called the "Workflow Anchor". The component Tasks in a Workflow are defined by the WS. These Tasks must be initiated by a Proxy. The initiator Proxy for all Tasks in the Workflow is the Workflow Anchor.

The need for an Anchor arises because once a Workflow has been branched into Tasks, other downstream entities don't have the complete picture of the Workflow and they cannot own the Workflow as a whole. The Workflow Anchor owns the Workflow. It is responsible for (a) executing the right tasks, (b) executing these tasks in the correct order (c) correctly accounting for tasks after execution, (d) handling failures in the right way when they arise, etc.

```
                                                 +--------+
                                         +-------| TASK-1 |
                                         |       +--------+
                         +----------+    |
             WORKFLOW    | Workflow |    |       +--------+
          -------------->|  Anchor  |----+-------| TASK-2 |
                         +----------+    |       +--------+
                                         |
                                         |       +--------+
                                         +-------| TASK-3 |
                                                 +--------+
```

Figure-7: Workflow Branching into Tasks

The Workflow Anchor can be located at multiple points in the network, such as the Client, Customer's Proxies, Provider's Proxies, etc.

For instance, the Client of services can be the Anchor and in this case the Client will have to own the workflow execution, service accounting and failure handling. The Customer's Proxies and Provider's Proxies may inspect or authenticate the messages in transit but they may have no understanding of the complete sequence of tasks, and will not be able to validate if the Client is executing the right sequence of actions. If the Client is unreliable, it may skip service accounting. The Client may perform invalid operations which may not lead to usable services, and then not account for these. This shows the importance of Anchor placement, and the trust relationship between the Provider and the Workflow Anchor.

To address the trust issues, we might decide to Anchor the Workflow in the Customer's Proxy or the Provider's Proxy. Now, the Client must request the Workflow execution from one of these Proxies. The Client should refer to the Workflow through some Workflow Name in order for the receiving Proxies to validate if the request is correctly formed. Now, the Workflow execution is owned by the Proxies (although the Client may know the Workflow composition and can frame the Workflow request). The Proxy will validate the Workflow before branching Tasks. The Proxy will ensure accounting and failure handling.

To execute any Workflow, the Workflow Anchor must be trustworthy. Depending on the deployment scenario, the Anchor may be situated at various points in the network. For personal clouds, the Client may be a valid Anchor. For private enterprise clouds, the Customer's Proxy may be a valid Anchor. For public clouds, the Provider's Proxy may be the only trustworthy anchor. For community clouds, the end Service Node may be the trustworthy anchor. Depending on where the Anchor is located, Workflow branching will take place at the Anchor.

```
+--------+        +----------+         +----------+       +---------+
| Client |        | Customer |         | Provider |       | Service |
|        |<------>|  Proxy   |<-------->|  Proxy   |<----->|   Node  |
+--------+        +----------+         +----------+       +---------+

 Personal           Private             Public            Community
 Cloud              Cloud               Cloud             Cloud
 Anchor             Anchor              Anchor            Anchor
```

Figure-8: Possible Workflow Anchor Locations

Network elements that are upstream from the Anchor MUST NOT be able to branch the Workflow into Tasks. These elements may validate the Workflow but they are not responsible for doing so. The Anchor MUST be responsible for validating the Workflow and correct execution. There SHOULD be an interface between Clients and Proxies, and between Proxies, through which they can request complete Workflows.

The Workflow request received at the Anchor may be incomplete. The request may specify parameters about a VM, and may leave the details of network, storage and security to the Anchor. The request received at the Anchor is treated as indicative by the Anchor. The Anchor SHOULD forward the received request to the WS and obtain a complete and accurate description of the Workflow prior to executing it.

6.2.4. Distributed Workflow

It should be possible to compose Workflows by combining Workflows. Each of these Workflows and their combinations may have its own (different) Anchor. Each Anchor will completely own its Workflow, and a higher level Anchor can own a Workflow combination.

For example, a customer may create a complex service by combining Workflows in a private cloud and a public cloud through a single request. Or, one provider may create services in their network and another provider's network through the same request. Or, a single request may be orchestrated by multiple domain-specific proxies within a provider's network. To distribute Workflows across service domains and provider/customer boundaries, a large Workflow may be decomposed into individual Workflows owned by individual Proxies.

```
                    Workflow-M  +---------+
                 +--------------| Proxy-B |
    Workflow     |              +---------+
       MN        |                   |
       |         |                   V
       V         |                Execute
   +---------+   |              Workflow-M
   | Proxy-A |---+                               Workflow-Y +---------+
   +---------+   |                             +------------| Proxy-D |
                 |               Execute       |            +---------+
                 |              Workflow-X     |                 |
                 |                   ^         |                 V
                 |                   |         |              Execute
                 |  Workflow-N  +---------+    |             Workflow-Y
                 +--------------| Proxy-C |----+
                                +---------+    |              Execute
                                               |             Workflow-Z
                                               |                 ^
                                               |                 |
                                               | Workflow-Z +---------+
                                               +------------| Proxy-E |
                                                            +---------+
```

Figure-9: Distributing Workflows

In Figure-8, a workflow branching is illustrated. Proxy-A receives a two-stage workflow called MN (M and N are two stages) and forwards it to Proxies B and C. Proxy B executes the Workflow M as is. The Proxy C divides the Workflow N into Workflows X, Y and Z. Proxy C executes

the workflow X by itself, and delegates Y and Z to other proxies D and E. Workflow division allows intelligence related to a Workflow to be abstracted. The implementation of a Workflow may be changed while keeping the interface to it unchanged. The mapping between Workflows and their branching patterns may also be changed. This gives operators flexibility in how they want to deploy services.

7. Security Considerations

Encryption and authentication of SOP messages is described in the Protocol document [SOP]. The LP is responsible for securing a Provider's network. The LP and RP should establish secure connections over IPSec or other kinds of VPNs over the Internet.

8. IANA Considerations

NA.

9. Conclusions

This document described the architecture for separating service-independent and service-dependant orchestration functions at the network level. This architecture can be used to rapidly create new services and for security and policy control of service networks.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[NIST] DRAFT Cloud Computing Synopsis and Recommendations http://csrc.nist.gov/publications/drafts/800-146/Draft-NIST-SP800-146.pdf

[REQT] Service Orchestration Protocol Requirements http://www.ietf.org/id/draft-dalela-orchestration-00.txt

[XSD] XML Schema Description http://www.w3.org/XML/Schema

[OCL] Object Constraint Language http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL

11. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Authors' Addresses

Ashish Dalela
Cisco Systems
Cessna Business Park
Bangalore
India 560037

Email: adalela@cisco.com

Mike Hammer
Reston
Virginia
USA 20190

Email: mphmmr@gmail.com

Monique Morrow
Cisco Systems [Switzerland] GmbH
Richistrasse 7
CH-8304
Wallliselle n
Switzerland

Email: mmorrow@cisco.com

Peter Tomsu
Cisco Systems Austria GmbH
30 Floor, Millennium Tower
Handelskai 94-96
A-1200   Vienna
Austria

Email: ptomsu@cisco.com