ACTIVE AETHER

A NEXT-GENERATION ARCHITECTURE FOR WEB SERVICES ON THE INTERNET

Robert F. MacInnis, Ph.D.
June 2017

AetherWorks
501 Fifth Avenue
New York, NY 10017

E: info@activeaether.com

1 INTRODUCTION

The following defines each component of the next-generation architecture described. It begins by introducing the core concepts and abstractions in the architecture and presents an overview of its structure. The roles and responsibilities of each actor in the architecture are defined next, together with their descriptors and the registries used to hold these descriptors. The framework for interaction between each actor is detailed last, together with deployment and invocation use-cases.

The term 'model' should be understood to mean 'architectural model'; the term 'infrastructure' is used to refer to the abstract notion of a 'running' architecture – that is, a collective reference to the described architectural components existing in an operational state, independent of any particular implementation.

2 OVERVIEW

The architecture described takes an end-to-end or 'holistic' approach to addressing the previously-identified shortcomings of the traditional Web Services model. The architecture presents a multi-endpoint Web Service environment which abstracts over Web Service location and technology and enables the dynamic provision of highly-available Web Services. The model describes mechanisms which provide a framework within which Web Services can be reliably addressed, bound to, and utilized, at any time and from any location.

The presented model aims to ease the task of providing a Web Service by consuming deployment and management tasks. It eases the development of consumer agent applications by letting developers program against what a service does, not where it is or whether it is currently deployed. It extends the platform-independent ethos of Web Services by providing deployment mechanisms which can be used independent of implementation and deployment technologies. Crucially, it maintains the Web Service goal of universal interoperability, preserving each actors' view upon the system so that existing Service Consumers and Service Providers can participate without any modifications to provider agent or consumer agent application code. Lastly, the model aims to enable the efficient consumption of hosting resources by providing mechanisms to dynamically apply and reclaim resources based upon measured consumer demand.

2.1 Providing Web Services

The presented model addresses the goal of reducing complexity for participants in the Web Service lifecycle by partitioning the responsibility of providing a Web Service into multiple independent roles, reducing the amount of domain-specific knowledge required by each actor and lowering the barriers to participation in the provision of Web Services. This process begins by treating the tasks of publishing, deploying and hosting a Web Service as distinct, independent activities. The traditional Service Provider role is thus decomposed into three autonomous actors: *Publisher*, *Manager*, and *ServiceHost*. In collectively fulfilling the responsibilities of the traditional role of Service Provider, these actors are supported by two new architectural entities: repositories called the *ServiceLibrary* and *HostDirectory*. These actors and entities are shown in Fig. 1 below together with labeled edges indicating the abstract interactions carried out between them. Each of these interactions is covered in the more detail in the paragraphs to follow.

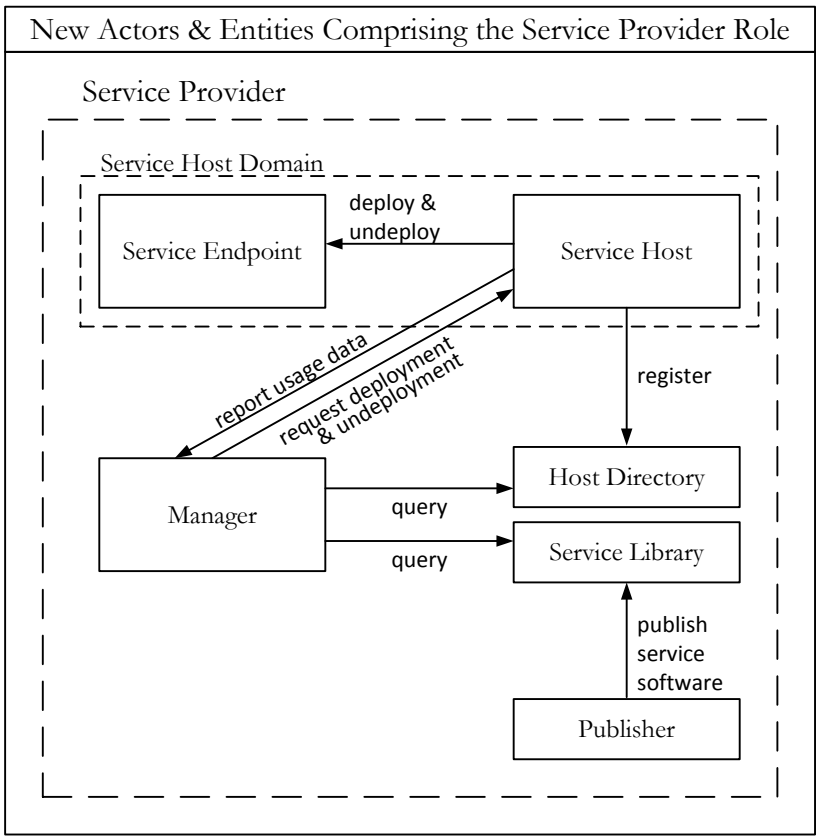

Fig. 1: Interactions between the new actors of ServiceHost, Manager and Publisher, and the two new architectural entities of HostDirectory and ServiceLibrary which together provide the functionality of the traditional Service Provider role.

Rather than being deployed explicitly, a Web Service provider agent implementation is instead described by a Publisher who then 'publishes' it into the infrastructure by storing it in a repository called the

ServiceLibrary. This approach represents a break from the traditional approach to Web Service deployment by separating service substantiation from actual realization: in the presented model, the lifecycle of a Web Service begins when it is published, not when it is deployed.

In order to participate in the infrastructure, ServiceHosts register their willingness to host Web Services by describing their available resources and registering with a directory called the HostDirectory. ServiceHosts indictate their available Web Service deployment containers and specify the list of Publishers whose Web Service provider agent implementations they are willing to deploy. ServiceHosts thus participate in an infrastructure not by advertising their statically deployed services, but by advertising their hosting capabilities, joining a shared pool of latent hosting resources which can be dynamically consumed (and reclaimed) by Managers as necessary to meet changing levels of demand.

Managers are responsible for managing the provisioning level of a single Web Service (for which there may be zero or more endpoints at any given time). In order to enact deployment, Managers first query the ServiceLibrary and HostDirectory, then create deployment plans by pairing Web Service implementations with a suitably capable ServiceHost. Managers send deployment requests to ServiceHosts who are then are responsible for instantiating an endpoint of the Web Service (or denying the request). ServiceHosts provide information about the usage of each Web Service endpoint deployed within their domain to each Web Service's Manager. A Manager can use this usage data to make decisions about the necessary level of provisioning of the Web Service they manage.

2.2 Activating the Discovery Service Role

The responsibilities of the traditional Discovery Service role are consumed by a new actor called the *ActiveServiceDirectory*. The ActiveServiceDirectory holds mappings between a single Web Service and a set of active endpoints of the Web Service. A central entity in the architecture, the collective mappings held in the ActiveServiceDirectory represent the current state of an infrastructure from all participants' perspectives. Because all infrastructure participants rely on the ActiveServiceDirectory to locate endpoints of their desired Web Services, this directory is ideally placed to instigate autodeployment procedures for Web Services that have no currently deployed endpoints.

The ActiveServiceDirectory provides operations to add, remove and locate active endpoints of Web Services. If the ActiveServiceDirectory receives a lookup request for the active endpoints of a particular Web Service, but no such entries exist, the ActiveServiceDirectory is responsible for proactively locating and informing

the Manager of the requested Web Service. The Manager can dynamically initiate the deployment of a new endpoint using the previously described deployment procedure. Managers are responsible for inserting and maintaining all ActiveServiceDirectory entries for any newly deployed endpoints of the Web Service it manages. If demand for its Web Service drops to zero, a Manager may decide to undeploy one (or all) of the deployed endpoints. If an endpoint is undeployed the Manager removes the endpoint entry from the ActiveServiceDirectory, ensuring that the directory remains up-to-date and as accurate a reflection as possible of the current state of the infrastructure. The interactions between the ActiveServiceDirectory (as Discovery Service) and Managers are shown in Fig. 2 below.

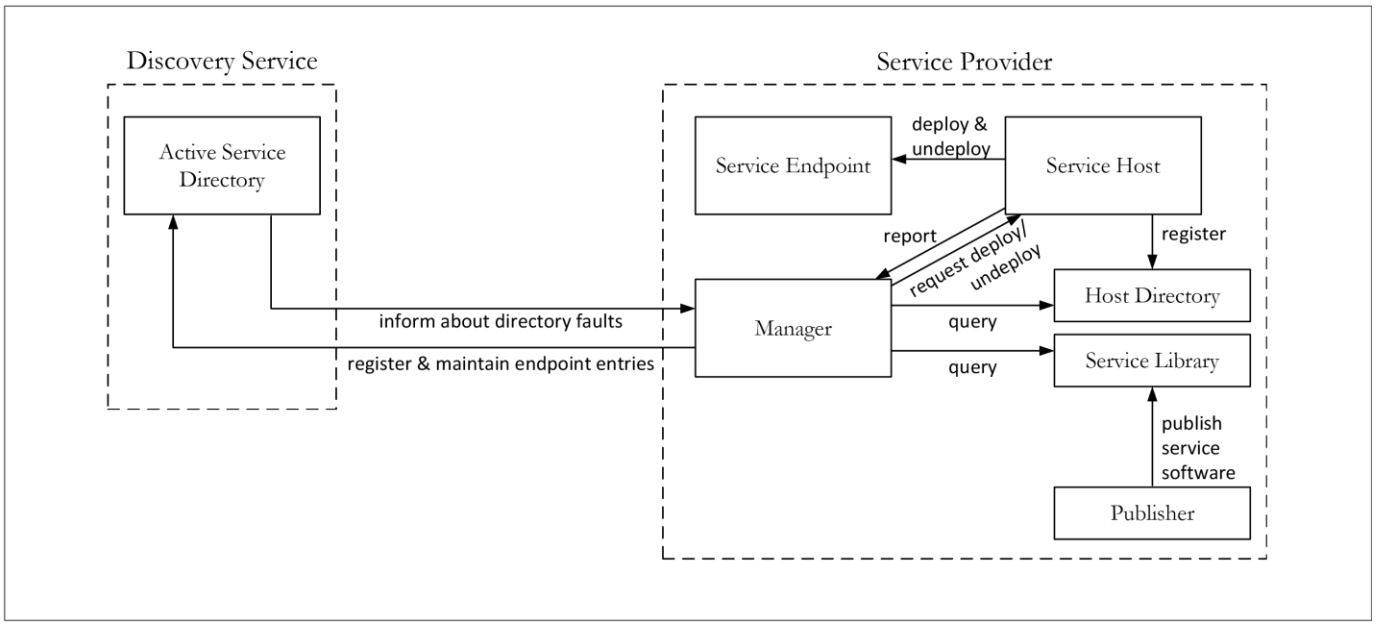

Fig. 2: The interactions between the ActiveServiceDirectory (as Discovery Service) and the actors & entities comprising the Servicer Provider role.

2.3 Consuming Web Services

The presented architecture does not use URLs to describe Web Services since URLs may become invalid over time. Web Services are instead identified with a URI, abstractly describing a service which at any point in time may have zero or more active endpoints. The *ServiceConsumer* actor is relieved from the tasks of locating and binding to Web Service endpoints through the introduction of a mechanism which robustly performs these tasks on their behalf.

The architecture presents a limited-mediation framework for the consumption of Web Services which transparently resolves a live endpoint URL from the URI contained in an invocation request. This framework

is realized as proxy mechanism, residing at the Service Consumer, which acts as a gateway into an instantiation of the architecture – a 'point of presence', as shown in Fig. 3. Consumer agent software is written to bind to this local point of presence and invoke Web Service operations using the desired Web Service's URI. The point of presence is responsible for transparently resolving a URL from the URI by retrieving a list of active endpoints of the Web Service from the ActiveServiceDirectory, selecting an endpoint for use, invoking the requested operation on behalf of the Service Consumer, and returning any results. It is also responsible for transparently detecting and recovering from the failure of Web Services and the ServiceHosts on which they are deployed, and for proactively recovering from these failures by retrying alternative endpoints (according to local policy). Only unrecoverable errors are returned to Service Consumers, indicating that given the available resources of the infrastructure, it is not currently possible to fulfill the request.

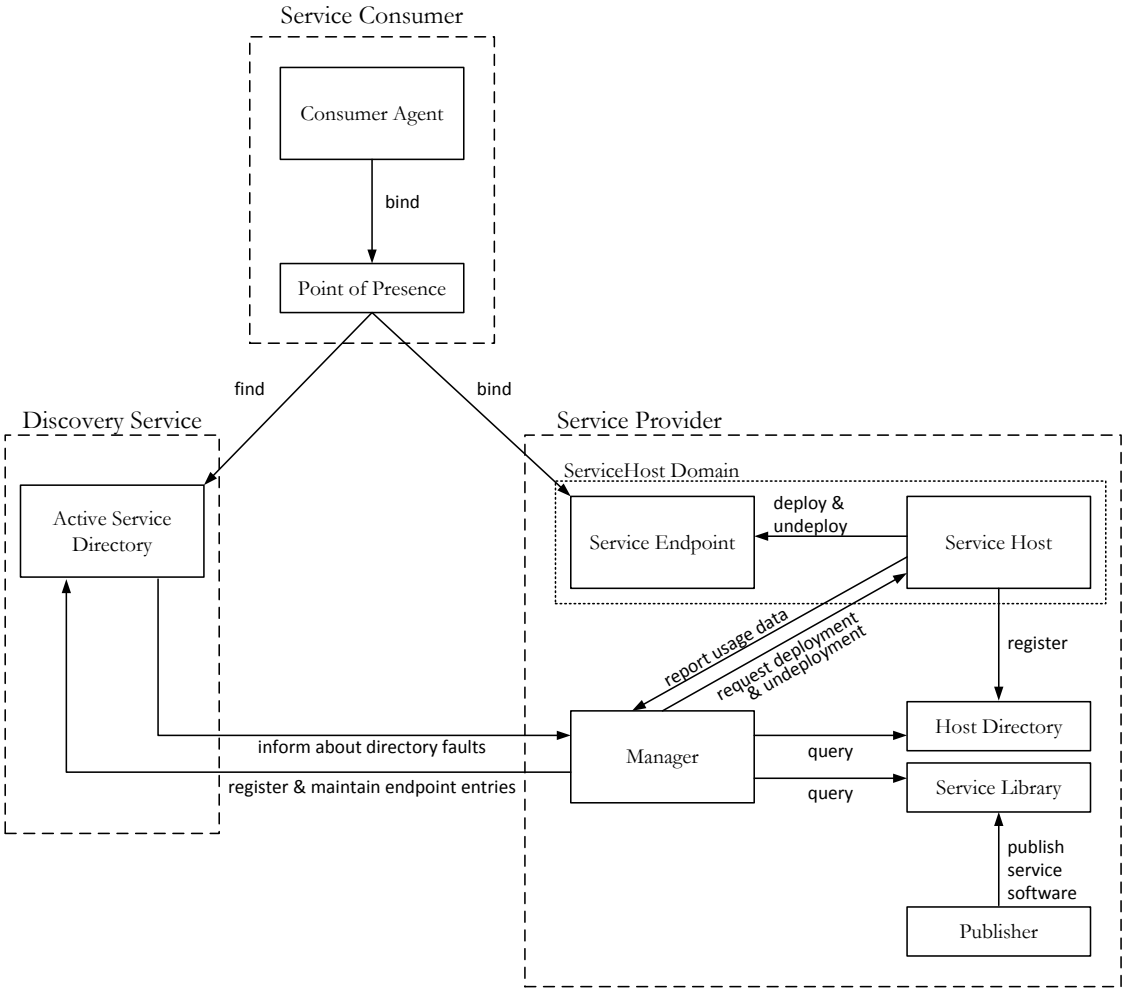

Fig. 3: Actors, entities and interactions in the newly presented architecture.

The point of presence abstracts over the location of a service while still preserving the current Service Consumer actor's view upon the system. It simplifies the creation of consumer agent applications by allowing developers to program against what a Web Service does, not where it is or whether it is currently deployed. Further, because it consumes all tasks which require interaction with the Discovery Service, the point of presence provides a layer of abstraction over the particular standards versions used in an infrastructure (e.g. UDDI version). This provides a barrier to obsolescence in the face of evolving standards while enabling consumer agent applications to be portable between environments which use different standards.

The process of transparently resolving endpoint URLs from the provided URI also factors out the redundant failure-detection and recovery code necessary in the traditional Web Services usage pattern. Multi-endpoint environments provide alternative methods of approximating these properties by offering an external single point of contact responsible for each individual Web Service. By moving the proxy indirection out of the infrastructure and onto the client, the presented architecture not only conserves resources, but removes the bottleneck and central point of failure introduced by remote proxies, assuming that if the local point of presence dies, the client has died too.

The diagram in Fig. 3 presents the structure and relationships between the newly introduced actors, with directed links indicating the actions carried out between them. The full responsibilities of each actor are detailed in the upcoming section, "Next-Generation Actors: Roles & Responsibilities".

3 NEXT-GENERATION ACTORS: ROLES & RESPONSIBILITIES

3.1 Publisher

The *Publisher* actor is responsible for bringing an implemented Web Service into a repository: it acquires the location of the provider agent code from a developer, describes its deployment container and hosting requirements using a ServiceImplementationDescriptor (shown in Fig. 4 below), and publishes it into a library of implementations called a *ServiceLibrary*.

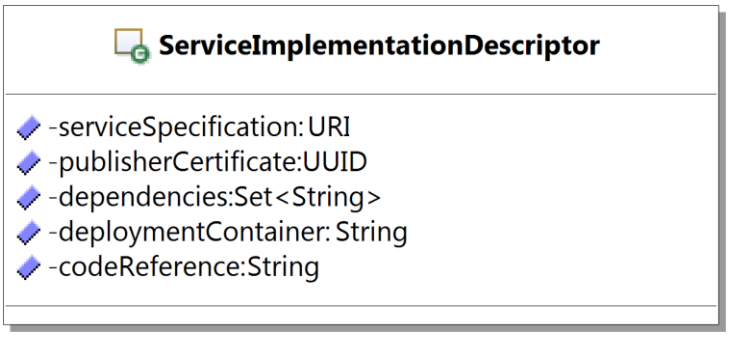

Fig. 4 The 'ServiceImplementationDescriptor' is used by a Publisher to identify a single implementation of a Web Service provider agent application.

A ServiceImplementationDescriptor is used to describe an implementation of a Web Service. It includes the URI of the implemented Web Service, a UUID identifying the Publisher, the location of the service code, a set of hosting environment requirements as Strings, and the String identifying the required deployment container. The uniquely-identifiable *PublisherID* identifies a single Publisher and can be used to guide decisions on service deployment and invocation-time endpoint selection. At any point in time there may be multiple published implementations of a Web Service, each published by a different Publisher, which implement the same Web Service (as identified by the 'serviceSpecification' element of the ServiceImplementationDescriptor). A major benefit of this approach is the increased likelihood of a Web Service implementation being compatible for deployment on one of the set of currently available hosts.

3.2 ServiceHost

ServiceHosts control one or more Web Service deployment containers which are capable of hosting and exposing Web Services. Each ServiceHost must expose a remotely-accessible Web Service interface for the provision of Web Service endpoints conforming to the interface 'IServiceHost' shown below in Fig. 5. ServiceHosts are required to fulfill deployment & undeployment requests received through this interface. For every Web Service endpoint deployed in their domain, ServiceHosts are required to record and periodically report endpoint usage data to the service's managing entity (described next in section 3.3 'Manager'). For each deployed Web Service, ServiceHosts report usage data at a frequency specified in the 'reportPeriod' parameter of the 'deployServiceInstance' operation. This data is used by the managing entity to make decisions on endpoint provisioning levels.

IServiceHost

```
ServiceInstanceDescriptor deployServiceInstance(
                          ServiceImplementationDescriptor sid,
                          long reportPeriod);
boolean undeployServiceInstance(ServiceInstanceDescriptor sid);
```

Fig. 5 The IServiceHost interface for managing the deployment and undeployment of Web Service endpoints.

Shown in Fig. 6 below, a *HostDescriptor* is used to describe a ServiceHost. Each HostDescriptor includes the ServiceHost's uniquely-identifiable *hostID*, a reference to the location of their IServiceHost Web Service, and a description of their hosting capabilities (such as the available deployment containers). In order to indicate their willingness to participate in the hosting of Web Services, a ServiceHost stores their HostDescriptor in a directory called the *HostDirectory*.

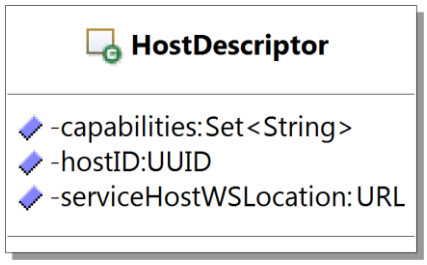

Fig. 6 A HostDescriptor describes the capabilities, location and identity of a single ServiceHost.

3.3 Manager

Managers are responsible for managing the provisioning level of a particular Web Service. Managers create and execute deployment plans in order to ensure that an adequate number of endpoints are deployed in order to meet the current level of demand for the Web Service being managed. Managers interact directly with ServiceHosts in order to request the deployment and undeployment of endpoints and are responsible for registering and maintaining Web Service endpoint entries in a directory called the *ActiveServiceDirectory* (described next in section 3.4).

IManager

```
long reportUsageData(String data);
ServiceInstanceDescriptor requestFirstInstance(String serviceURI);
```

Fig. 7 Each Manager must expose a Web Service implementing the IManager interface. The 'reportUsageData' operation is used by ServiceHosts to report usage data for the Web Service under management.

Each Manager is required to expose a remotely-accessible Web Service conforming to the *IManager* interface, shown in Fig. 7 above. ServiceHosts report usage data to Managers through the `reportUsageData` operation of the IManager interface; in return, Managers return ServiceHosts a numerical value indicating the length of time the ServiceHost should wait before next reporting. Decisions about endpoint provisioning can be made based on the usage data returned from the ServiceHosts which are currently hosting endpoints of the Web Service under management. Detailed later in section 6.4 Demand-driven Dynamic Deployment', the `requestFirstInstance` operation is invoked by the ActiveServiceDirectory in the event that it receives a lookup request for the list of all active endpoints of a particular Web Service and it finds that no endpoints are currently deployed.

3.4 ActiveServiceDirectory

The *ActiveServiceDirectory* realizes the Discovery Service role, providing a directory that maps from Web Service URI to a set of active endpoints of the Web Service, each described using *ServiceInstanceDescriptor* (shown in Fig. 8 below).

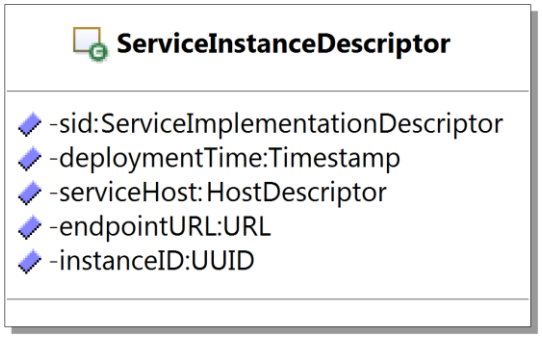

Fig. 8 A ServiceInstanceDescriptor describes a single deployed Web Service endpoint

A ServiceInstanceDescriptor describes a single deployed endpoint of a Web Service. It includes a HostDescriptor identifying the endpoint's ServiceHost, a ServiceImplementationDescriptor describing the implementation behind the deployed Web Service, the URL of the Web Service endpoint, and a timestamp indicating when it was deployed. These details can be used by Service Consumers to select a preferred endpoint from the set of all deployed instances of the desired Web Service. A unique instance identifier is also included in the descriptor and is used in situations where undeployment is requested on a Service Host with multiple instances of the same Web Service.

If there are no active endpoints of a Web Service the ActiveServiceDirectory is responsible for initiating the autodeployment of a new endpoint by contacting the Manager of the Web Service. By proactively addressing this 'directory fault', the ActiveServiceDirectory provides a crucial mechanism for efficient resource utilization by allowing for endpoints to be deployed in response to demand and remain 'available' without any pre-provisioned capacity. As detailed in the next chapter, Managers themselves are realized as Web Services and are published and deployed using the same mechanisms as the Web Services they manage. When demand drops to zero and a Manager undeploys the final endpoint of the Web Service, the Manager will no longer receive data from the Service Hosts on which the endpoints were previously deployed. Due to lack of demand (this time from Service Host to Manager) the Manager itself will be undeployed and the newly-released hosting capacity added back to the shared resource pool. This recursive management model enables an infrastructure with no demand for its Web Services to progressively wind down its deployments until it reaches a level of zero resource consumption.

3.5 Point of Presence

Service Consumers bind to and invoke Web Service operations on a local entity called the Point of Presence (POP). The POP is a transparent indirection mechanism – a proxy – that interacts with the infrastructure on behalf of the Service Consumer. When it receives an invocation request from a consumer agent application, the POP is responsible for using the ActiveServiceDirectory to resolve an active endpoint of the requested Web Service, invoking the requested operation on the endpoint, and returning any results to the requesting consumer agent application. It is responsible for transparently detecting and recovering from the failure of Web Service endpoints and the hosts on which they are deployed.

4 ARCHITECTURAL ENTITIES

4.1 Service Library

The Service Library serves as a repository for Web Service implementations. The Service Library provides a mapping from Web Service URI to the set of ServiceImplementationDescriptors describing implementations of the Web Service, each of which may be written in a different programming language for a different target deployment container. Publishers use the Service Library to publish and unpublish Web Service implementations. Managers use the lookup operations of the Service Library during the Planning phase in order to select an implementation for deployment on a Service Host.

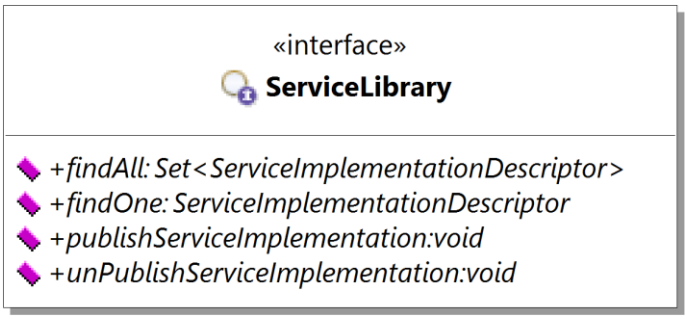

Fig. 9 The ServiceLibrary interface through which Publishers and Managers interact with the repository.

4.2 Host Directory

The HostDirectory is a repository which holds a mapping from PublisherID to a set of HostDescriptors describing the ServiceHosts registered as willing to host Web Services from the specified Publisher. Managers use the Host Directory during the Planning stage of deployment in order to select a Service Host on which to deploy a new Web Service endpoint.

Fig. 10 The HostDirectory interface is used by ServiceHosts and Managers to interact with the HostDirectory repository.

5 INTRA-ARCHITECTURE INTERACTIONS

5.1 Service Consumer to Local Point of Presence (POP)

Service Consumers do not interact directly with Web Service endpoints. In order to use a Web Service a Service Consumer always binds to and invokes operations upon an entity known as the 'local point of presence', shown in Fig. 11. POPs are co-located with Service Consumers; consumer agent software is written to statically bind to the URI of a Web Service, prefixed with the protocol, hostname and port of a well-known local URI (e.g. http://localhost/WebServiceURI) – thus creating a URL as per the W3C standard.

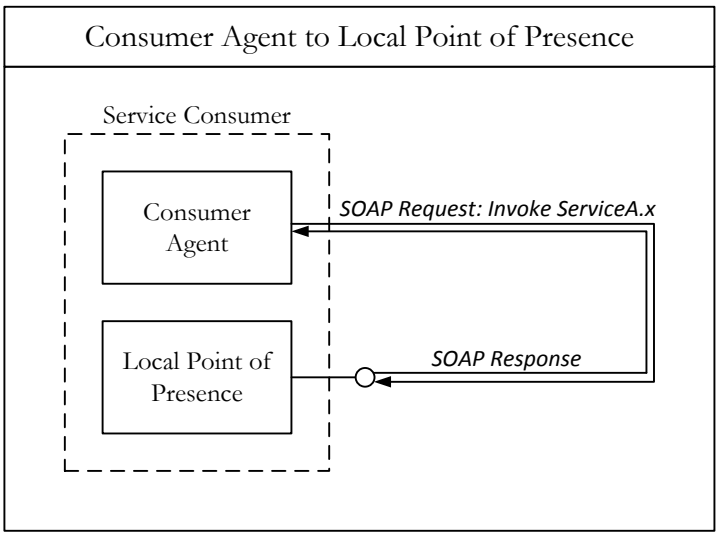

Fig.11 Consumer Agent binding to and invoking Web Service operations on the local point of presence.

The local point of presence entity transparently locates, binds to, and invokes the operations of Web Services on behalf of the Service Consumer (shown in Fig. 12 below). It is responsible for detecting and attempting to recover from the failure of Web Service endpoints and the hosts on which they are deployed (between Fig. 12 steps 5 and 7). As with all architectural entities, the local point of presence exerts a 'best effort' to complete the requested operation. All errors returned by the local point of presence are non-recoverable, indicating that, given the currently available resources of the system, it is simply not possible to carry out the requested operation.

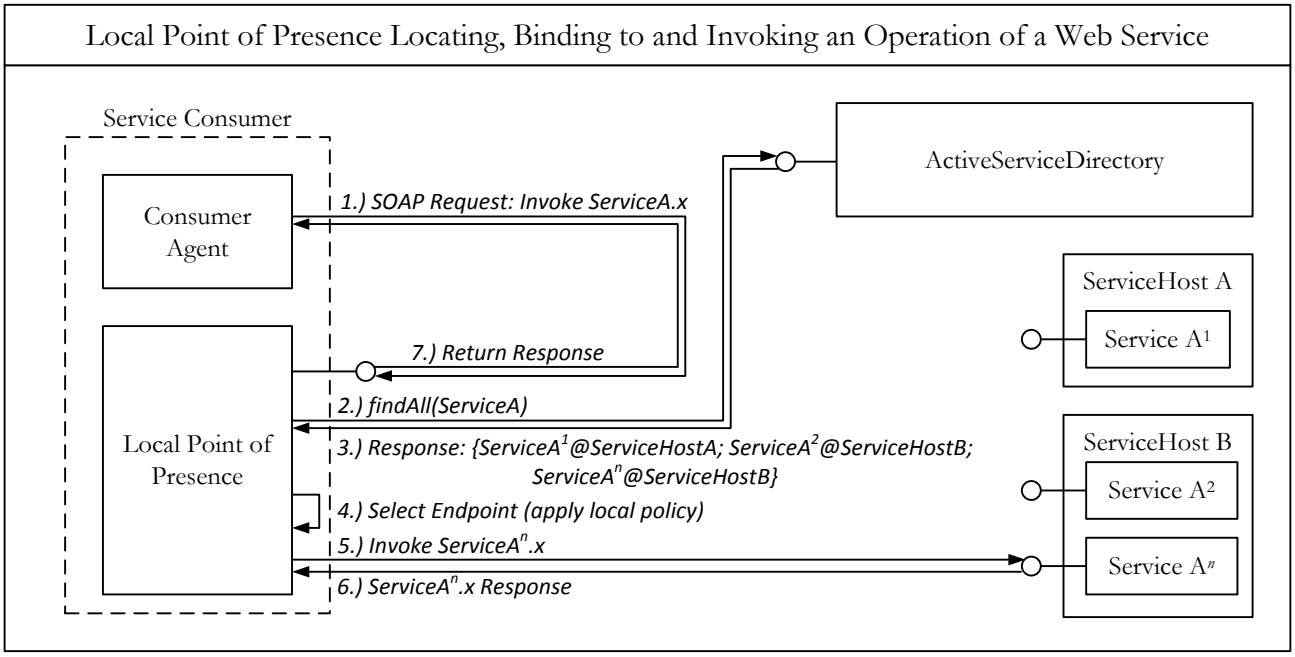

Fig. 12 The seven basic steps for invoking a Web Service operation via the local Point of Presence

The diagram in Fig. 12 above outlines the process of a Service Consumer invoking operation 'x' of the Web Service identified by the URI 'ServiceA'. The consumer agent application binds to and invokes the operation on the POP (step 1), which then must locate an endpoint of the requested Web Service. The POP sends a lookup request to the ActiveServiceDirectory by invoking its '`findAll`' operation with the URI 'ServiceA' as a parameter (step 2) and receives back a list of ServiceInstanceDescriptors describing the currently active endpoints of ServiceA (step 3). The POP applies local policy to select which endpoint to use (step 4) before connecting to the endpoint and invoking operation 'x' on behalf of the Service Consumer (step 5). The results of the operation are returned to the POP (step 6) and finally returned to the consumer agent application (step 7).

It is important to note that Service Consumers, as commonly understood, are not the only entities in the architecture which use the facilities provided by a local point of presence. Entities commonly considered to be solely provider agents – such as Managers and Service Hosts – also use a local point of presence to carry out intra-architecture interactions. By factoring out the location and failure recovery mechanisms of both intra- and extra-architecture Web Service interactions, both consumers and providers of Web Services are freed from the responsibility of implementing their own proprietary mechanisms for these tasks. Thus a single application of effort toward the development of a reliable, robust, and efficient point of presence implementation will benefit all actors and entities which both use *and provide* the infrastructure.

5.2 Publisher to Service Library

Publishers are Service Consumers who interact with the ServiceLibrary in order to publish and unpublish implementation of a Web Service. As with all Service Consumers in the architecture, Publishers do not interact directly with any external Web Services, instead invoking the desired operations via the local point of presence.

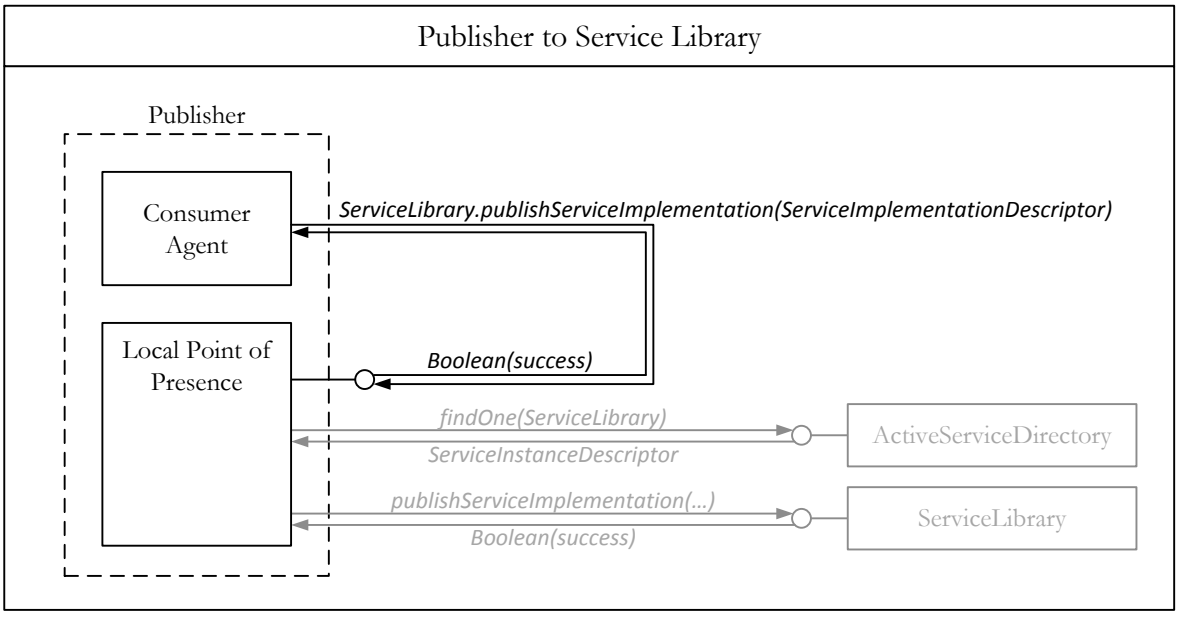

Fig. 13 Interaction between a Publisher and the ServiceLibrary in order to publish a Web Service implementation.

In order to publish an implementation of a Web Service (described with a ServiceImplementationDescriptor), Publishers bind to the local point of presence and invoke the 'publishServiceImplementation' operation of the ServiceLibrary with the desired ServiceImplementationDescriptor, as shown in Fig. 13 above. Once an implementation has been published it becomes immediately available for use, able to be deployed onto capable ServiceHosts as necessary to meet demand.

5.3 Service Host to Host Directory

While ServiceHosts contribute to the role of Service Provider, they may also act as Service Consumers. In order to register themselves with the Host Directory, ServiceHosts first describe themselves using a HostDescriptor, bind to the local point of presence, and invoke the 'addAuthorizedPublisher' operation of the HostDirectory, as shown in Fig. 14.

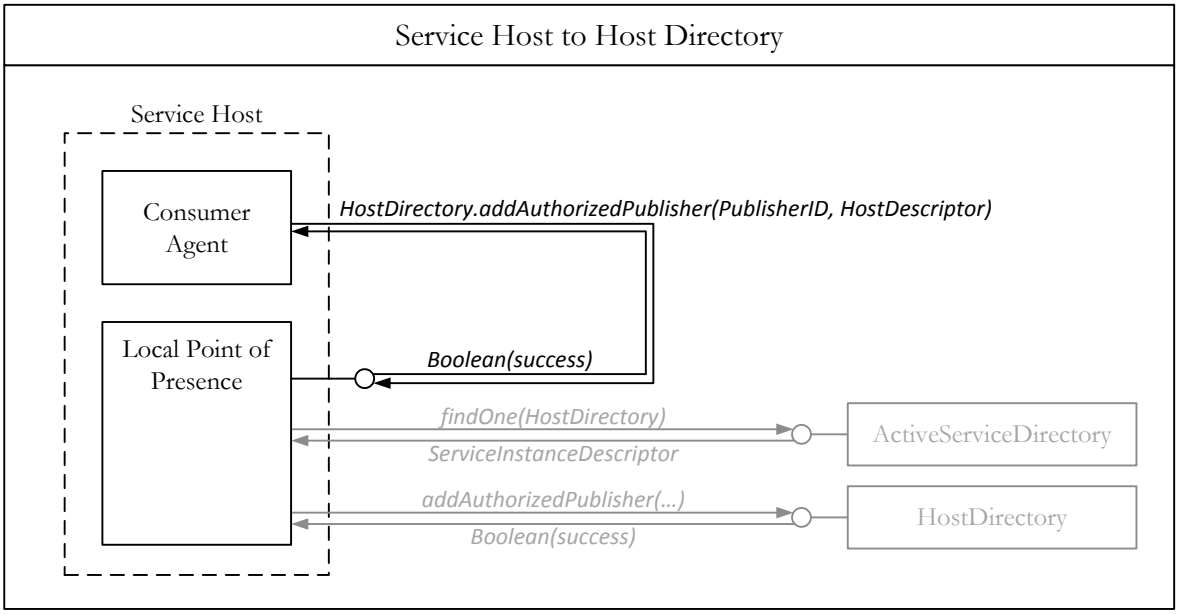

Fig. 14 ServiceHosts register their HostDescriptor with the HostDirectory, indicating the Publishers whose Web Service implementations they are willing to deploy and host.

By operating through the local point of presence the ServiceHost registration process is thus performed with the same robust invocation procedures provided by the infrastructure to all Service Consumers. Upon registration, a Service Host's HostDescriptor is added to a shared pool of hosting resources, ready to be consumed as necessary to meet demand.

5.4 POP to Active Service Directory

The point of presence is a transparent endpoint resolution and failure recovery mechanism which locates, binds to, and invokes operations upon Web Services on behalf of the Service Consumer. In order to locate an active instance of a Web Service the point of presence interacts with a Web Service called the 'ActiveServiceDirectory', shown in Fig. 15 below, invoking either its '`findOne`' or '`findAll`' operations with the URI of the desired Web Service as a parameter.

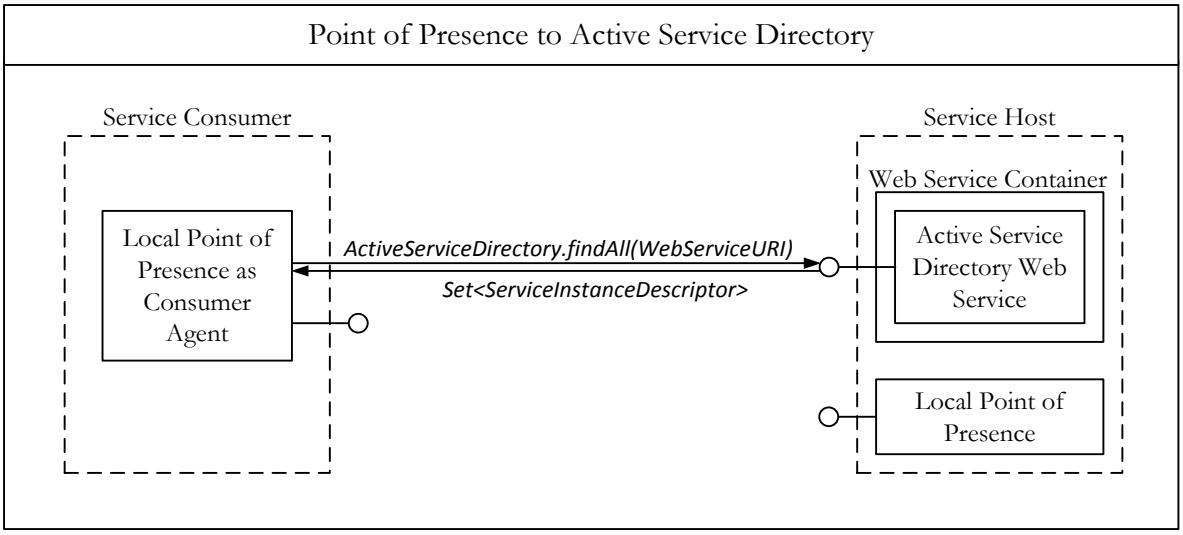

Fig. 15 Point of Presence interacting with the ActiveServiceDirectory in order to retrieve a set of all known endpoints of a Web Service

In order to participate in an instantiation of the architecture, the local point of presence must have a-priori knowledge of at least one ActiveServiceDirectory endpoint. Any alternative ActiveServiceDirectory endpoints are listed in the ActiveServiceDirectory under the Web Service URI 'ActiveServiceDirectory' and can be located using the same `findOne` and `findAll` operations. It is recommended that the POP periodically retrieve and store a list of alternative ActiveServiceDirectory endpoints to use in the event of failure.

5.5 ACTIVE SERVICE DIRECTORY TO ACTIVE SERVICE DIRECTORY

When the ActiveServiceDirectory receives a lookup request for a Web Service with zero active endpoints, it must locate an instance of that Web Service's Manager so that an endpoint may be deployed. The ActiveServiceDirectory utilizes its own lookup facilities by binding to its local point of presence and invoking the ActiveServiceDirectory '`findOne`' or '`findAll`' operation using the URI of the Managing entity responsible for the originally requested Web Service. This URI is constructed by concatenating the well-known 'Manager' URI to the URI of the requested Web Service in the pattern 'WebServiceURI_ManagerURI' (shown below in Fig. 16). The ActiveServiceDirectory is returned a ServiceInstanceDescriptor describing a Web Service endpoint which implements the 'IManager' interface described earlier.

Although the ActiveServiceDirectory is looking up a service in the ActiveServiceDirectory, there are no restrictions on how the ActiveServiceDirectory Web Service may be implemented, and thus no guarantee that the required service listing will be held by the requesting ActiveServiceDirectory. For this reason it is important that the operation be invoked using the POP, rather than the ActiveServiceDirectory simply looking in its local map.

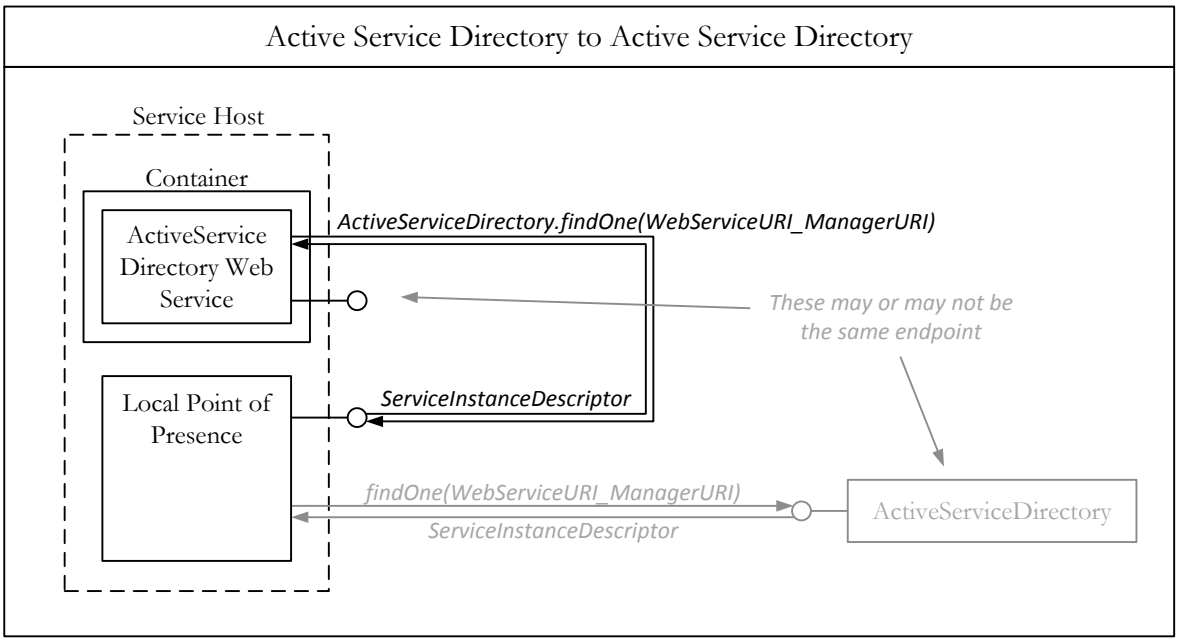

Fig. 16 ActiveServiceDirectory invoking the findOne operation of the ActiveServiceDirectory via the local POP

5.6 ACTIVE SERVICE DIRECTORY TO MANAGER

In order to request deployment of the first endpoint of a Web Service, the ActiveServiceDirectory invokes the 'requestFirstInstance' operation of the requested Web Service's managing entity. As shown in Fig. below, the URI of the requested Web Service is used as a parameter of this call which returns a boolean value indicating whether deployment was successful.

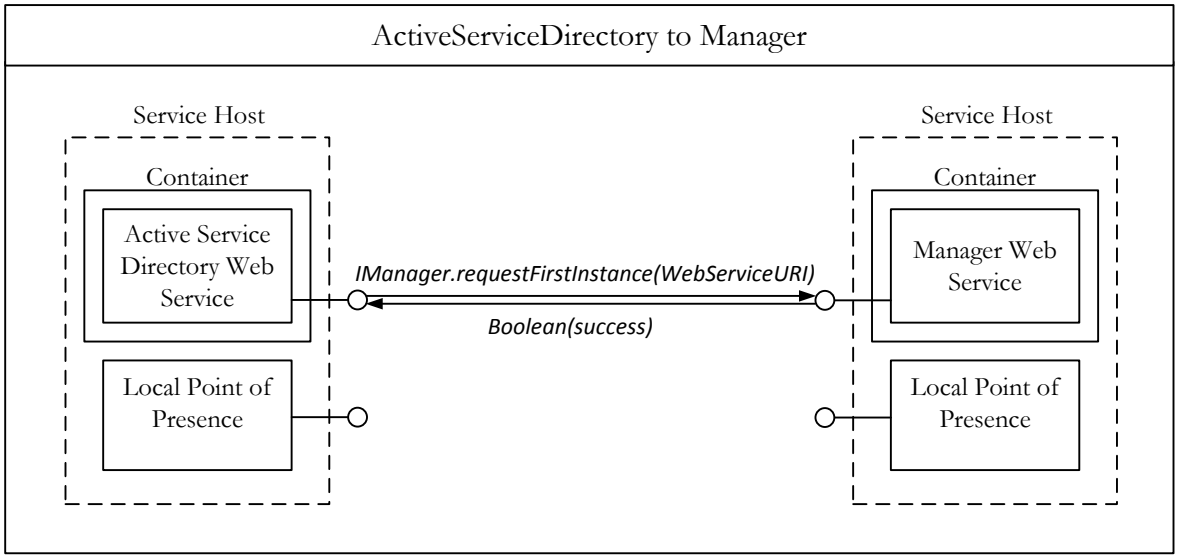

Fig. 17 The ActiveServiceDirectory invoking the 'requestFirstInstance' operation of an IManager Web Service.

As a Manager is responsible for maintaining the directory records of the Web Service they manage, Managers must insert a record of any endpoint they deploy into the ActiveServiceDirectory. Managers perform this operation in-band with the 'requestFirstInstance' invocation – returning 'true' if the entire deployment and registration process completed successfully, or 'false' otherwise. If the registration procedure is instead performed out of band there is a risk of a 'resource leak': a Manager returning 'true' after deployment but failing to register the new endpoint in the ActiveServiceDirectory will cause the newly deployed endpoint to become 'stranded', permanently consuming resources without being of use.

The 'requestFirstInstance' operation returning 'false' indicates that, given the current state of the system, it is simply not possible to deploy a new instance of the Web Service. This situation represents an unrecoverable error which is returned to the requesting entity (i.e. the Service Consumer). If the operation returns 'true' the ActiveServiceDirectory re-performs the originally-requested lookup operation and returns the newly-inserted ServiceInstanceDescriptor to the requesting entity.

5.7 MANAGER TO SERVICE LIBRARY

During the deployment process, Managers invoke the operations of the Service Library in order to retrieve a set of implementations of the Web Service being deployed (shown below in Fig. 18). These implementations are each described with a ServiceImplementationDescriptor; the elements of this descriptor can be used later by the Manager in order to select a suitable candidate implementation for deployment.

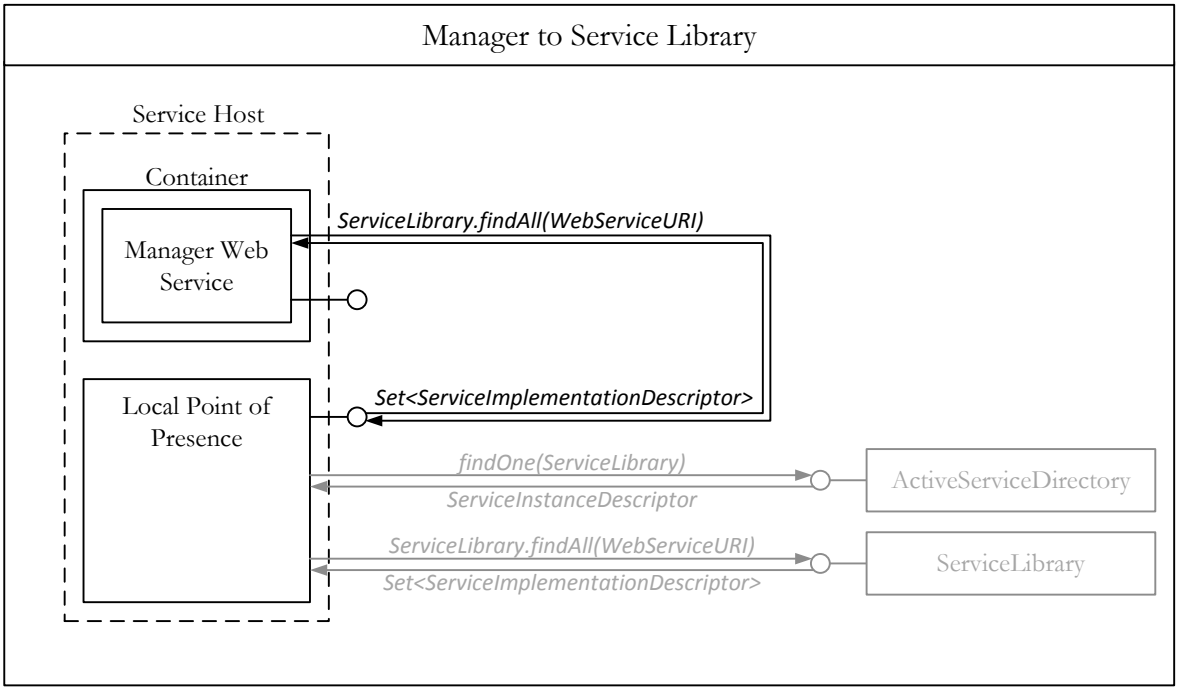

Fig. 18 Managers invoke the operations of the ServiceLibrary during deployment in order to retrieve the set of implementations of a Web Service.

5.8 MANAGER TO HOST DIRECTORY

Much like their interaction with the ServiceLibrary, Managers invoke the operations of the HostDirectory during the deployment process (shown in Fig. 19 below). Managers use the HostDirectory to retrieve a set of HostDescriptors identifying ServiceHosts willing to deploy Web Service implementations published by the indicated publisher (identified with a unique PublisherID). Managers compare the ServiceImplementationDescriptors retrieved from the ServiceLibrary with the HostDescriptors retrieved from the HostDirectory in order to craft suitable candidate deployment plans based on local policy. Once a plan is selected the Manager may contact the selected ServiceHost and initiate the deployment process, as detailed next.

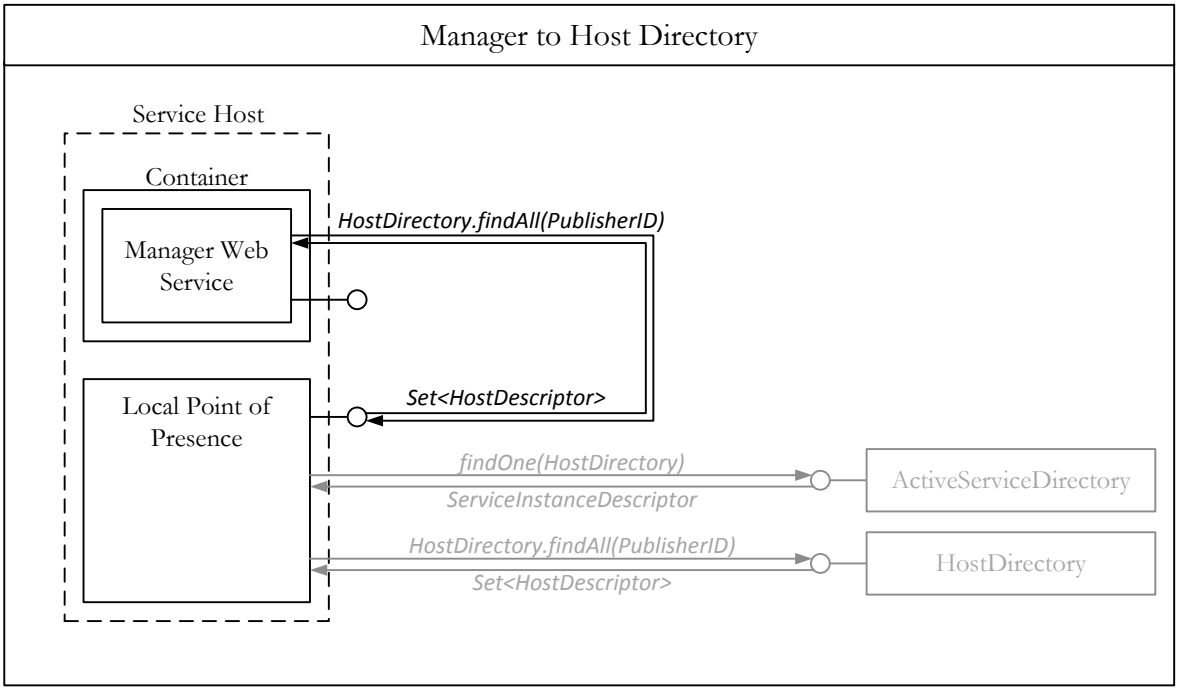

Fig. 19 Managers interact with the HostDirectory during the deployment process in order to retrieve a list of ServiceHosts willing to deploy Web Service implementations published by various Publishers.

5.9 Manager to Service Host

After selecting a suitable Web Service implementation (described with a ServiceImplementationDescriptor) for deployment on a selected Service Host (described with a HostDescriptor) a Manager binds to the ServiceHost included in the HostDescriptor and invokes the 'deploy' operation directly (shown below in Fig. 20). Upon successful deployment the Service Host returns the Manager a ServiceInstanceDescriptor describing the newly deployed Web Service endpoint. The Manager then registers this ServiceInstanceDescriptor in the ActiveServiceDirectory, completing the deployment process.

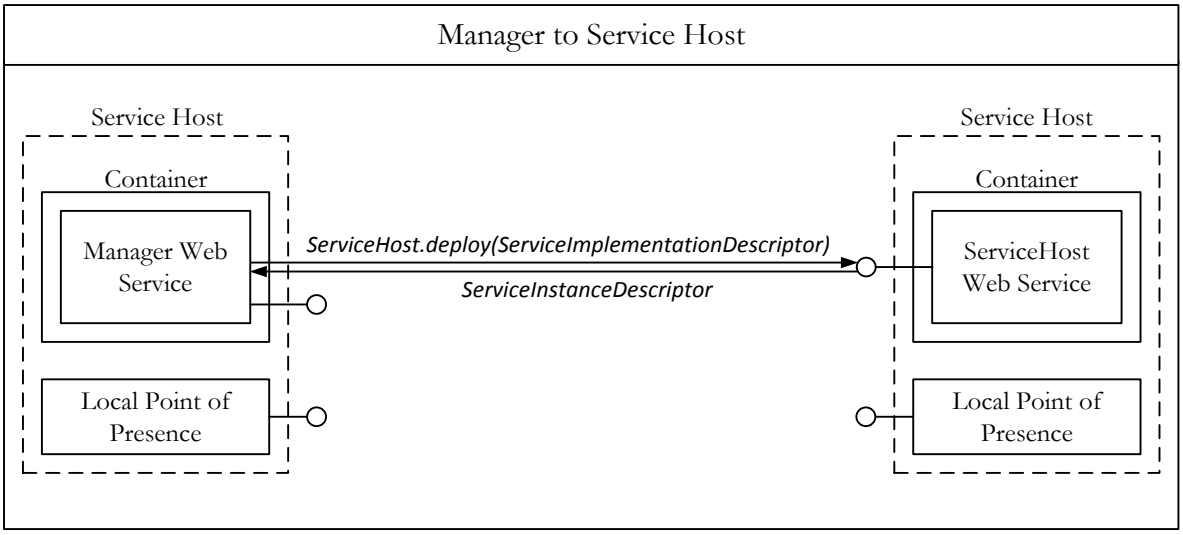

Fig. 20 Managers initiate the deployment of a Web Service endpoint on a ServiceHost by invoking the 'deploy' operation with a ServiceImplementationDescriptor describing the Web Service implementation to be deployed

5.9 Service Host to Manager

For each Web Service endpoint deployed within their domain (i.e. the containers under their control), a Service Host must report usage data to that Web Service's managing entity. Shown in Fig. 21 below, the Service Host binds to the local point of presence and invokes the '`reportUsageData`' operation of the Web Service identified by the concatenation of a well-known 'Manager' URI to the URI of the Web Service for which data is being reported (i.e. 'WebServiceURI_ManagerURI').

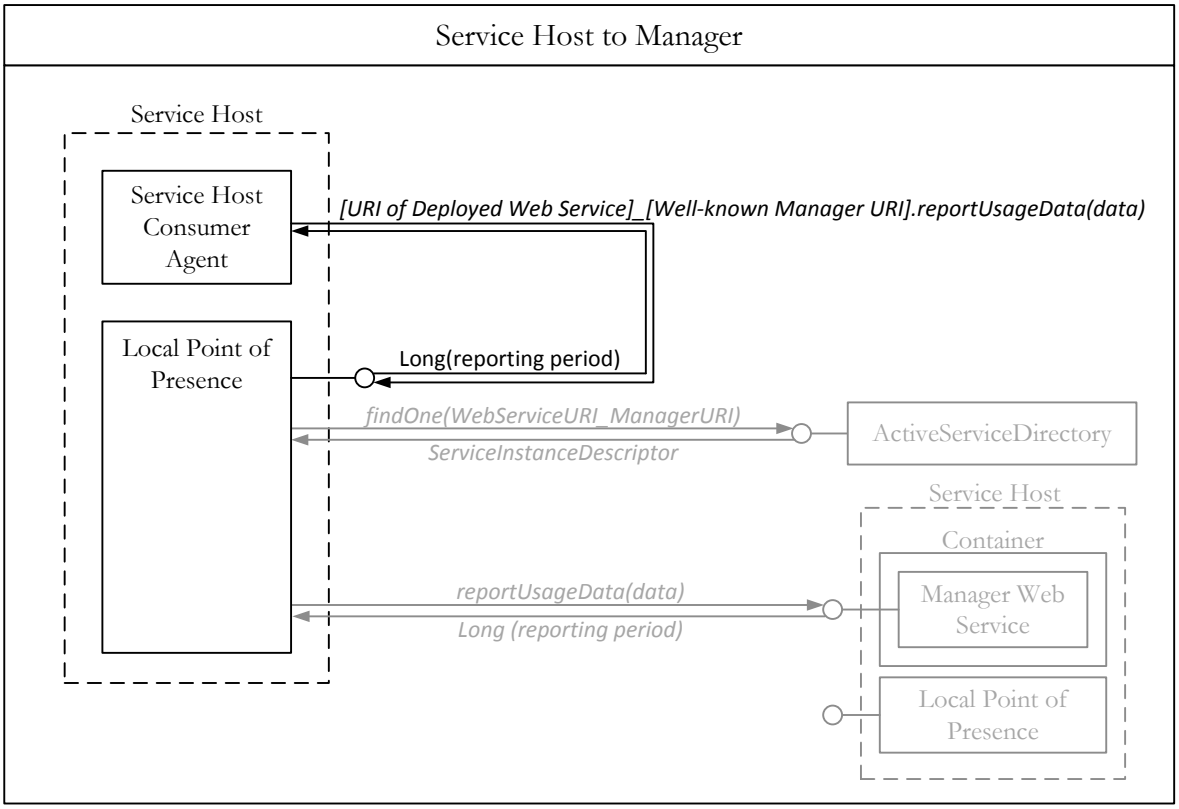

Fig. 21 ServiceHosts report usage data for each Web Service deployed within their domain by invoking the 'reportUsageData' operation of each Web Service's Manager.

The IManager '`reportUsageData`' operation returns a numerical value indicating the length of time the ServiceHost should wait before next reporting usage data for this particular Web Service. A ServiceHost's local policy may dictate that usage data be returned earlier than requested (e.g. due to local resource constraints, such as working memory); returning data significantly later than the requested period may indicate to the Manager that there is a problem with the ServiceHost – information which a Manager may act upon in order to effectively manage the availability of its service.

5.11 Manager to Active Service Directory

When a Manager of a Web Service successfully deploys or undeploys endpoints of that Web Service it must add or remove the endpoint reference in the ActiveServiceDirectory. Managers bind to the local point of presence and invoke the 'addActiveService' or 'removeActiveService' operation of the ActiveServiceDirectory Web Service using the relevant ServiceInstanceDescriptor (shown below in Fig.).

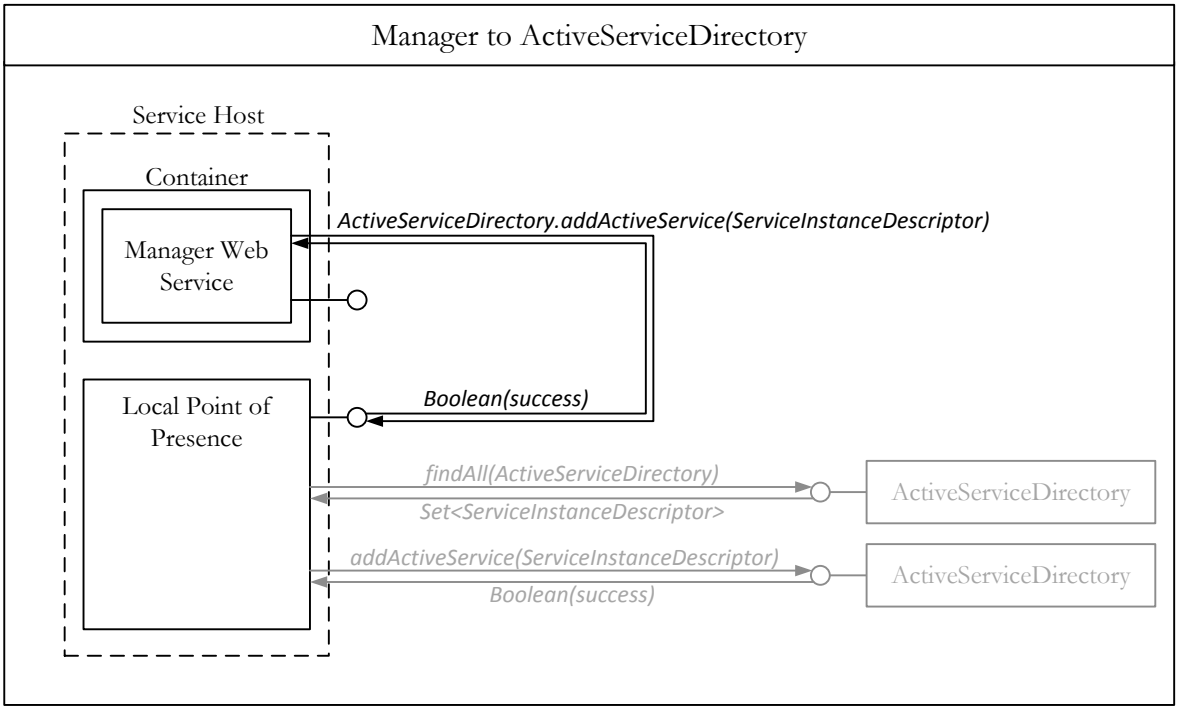

Fig. 22 Managers use the operations of the ActiveServiceDirectory to maintain the entries of the Web Service they manage.

Managers are responsible for maintaining the ActiveServiceDirectory entries of the Web Service they manage. Of all entities fulfilling the responsibilities of the Service Provider, Managers are deemed to be most interested in maintaining an accurate public record of the Web Service they each manage as it provides them with accurate usage data from which they may make more informed decisions in the fulfillment of their responsibility to manage the provisioning level of a Web Service. If the ActiveServiceDirectory entries are not maintained then active endpoints of the Web Service may not be available to Service Consumers (wasting resources due to idle deployments) and the effectiveness of load-balancing techniques will be diminished. Alternatively, if a list including many failed endpoints is returned to a ServiceConsumer's local point of presence, the requesting consumer agent application may be significantly slowed as the point of presence tries each failed endpoint in turn.

6 PROCEDURES

This section presents the high-level procedures of the architecture (e.g. deployment) by using the previously described intra-architecture interactions as single coarse-grain steps. The examples are presented in step-by-step walkthroughs of the procedures together with supporting diagrams. The following sub-sections detail the intra-architecture interactions involved in the processes of endpoint deployment, service

invocation, and demand-driven dynamic deployment, using the provision and invocation of a 'DayTime' Web Service as an example.

6.1 Endpoint Deployment

The deployment of a new Web Service endpoint is always carried out by a Manager. As shown in Fig. 23 below and described in the paragraphs to follow, the Manager interacts with the ServiceLibrary, HostDirectory, one or more ServiceHosts, and the ActiveServiceDirectory in order to make a new 'DayTime' Web Service endpoint available for use.

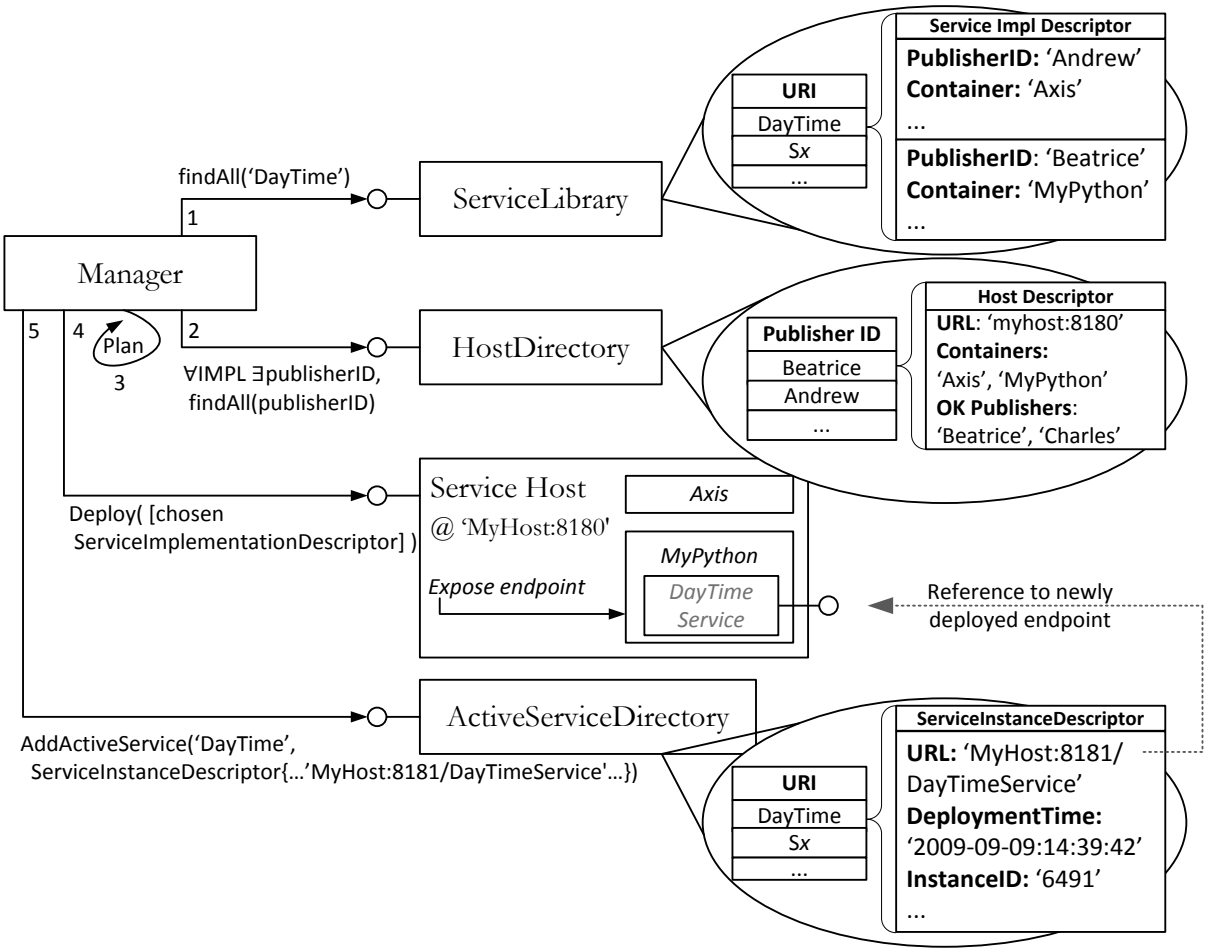

Fig. 23 Managers enact deployment by querying the ServiceLibrary and HostDirectory, creating a deployment plan, requesting deployment on a ServiceHost, and registering any newly deployed endpoint with the ActiveServiceDirectory.

In order to deploy a new Web Service endpoint a Manager first creates and selects a valid deployment plan (recall that a deployment plan consists of a ServiceImplementationDescriptor whose 'requirements' are matched by the 'capabilities' described in a HostDescriptor). Shown in Fig. 23, above, the Manager begins by

contacting the ServiceLibrary and retrieves the set of all published implementations of the 'DayTime' Web Service (Step 1). Each implementation returned is described with a ServiceImplementationDescriptor, each of which has a 'PublisherID' element. For each implementation, the Manager contacts the HostDirectory using the PublisherID (Step 2) and retrieves the set of ServiceHosts willing to deploy provider agent applications written by the specified publisher. The Manager then creates a set of valid deployment plans and selects one from amongst the candidates based on local policy (Step 3). Next the Manager executes the deployment plan, contacting the ServiceHost described by the selected HostDescriptor and invoking its 'deploy' operation with the selected ServiceImplementationDescriptor (Step 4). Upon successful deployment the target ServiceHost returns the Manager a ServiceInstanceDescriptor describing the endpoint. The Manager completes deployment by inserting the ServiceInstanceDescriptor into the ActiveServiceDirectory (Step 5). Once the newly deployed endpoint is listed in the ActiveServiceDirectory it is deemed to be deployed, ready to be located and its operations invoked by Service Consumers.

6.2 Service Invocation

Service Consumers bind to and invoke Web Service operations on the local point of presence. Once an invocation request is received, the local point of presence is responsible for locating an active endpoint of the target Web Service and robustly invoking the requested operation on behalf of the Service Consumer. The process of invoking the 'getTime' operation of the 'DayTime' Web Service is shown below in Fig. 24 and described in the paragraphs that follow.

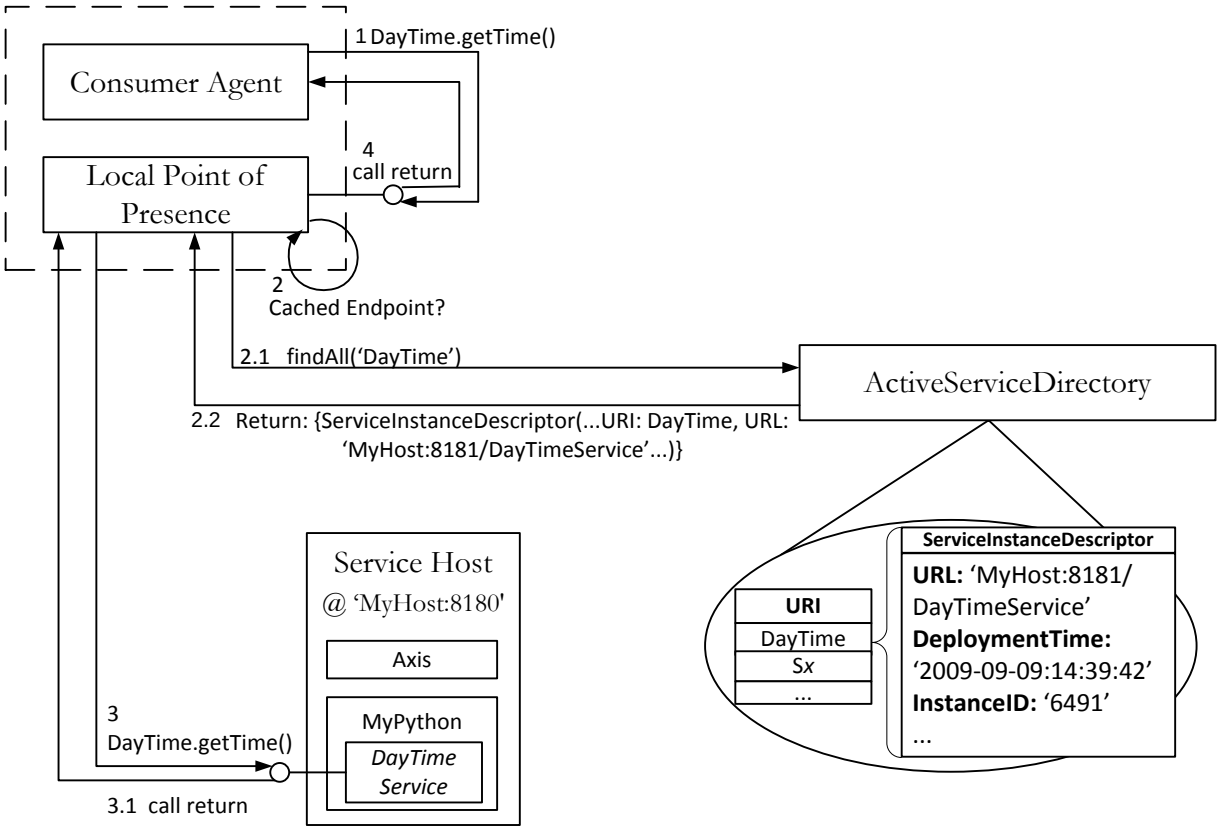

Fig. 24 The process of invoking the 'getTime' operation of the 'DayTime' Web Service using the Local Point of Presence, including location of active endpoints of the Web Service using the ActiveServiceDirectory, selecting an endpoint from the list of available endpoints, invoking the operation on a remote host, and finally return of the results to the Service Consumer

Upon receiving an invocation request from a consumer agent application (Step 1) the local point of presence first extracts the desired Web Service URI from the incoming request. If the point of presence implements an endpoint cache it may first check locally for a previously retrieved list of endpoints (Step 2). If no such entries exist, a new list must be retrieved using the ActiveServiceDirectory (Step 2.1). Once a list of endpoints is retrieved the point of presence selects one for use according to local selection policy (Step 2.2). The point of presence must then prepare the invocation request (including possible modifications to the message), bind to the selected Web Service endpoint, and forward the invocation request (Step 3). If the invocation fails for any reason the same procedure is attempted for the remainder of the active endpoints (Steps 2 and 3) and, if all endpoints prove unavailable, a generic error returned to the Service Consumer (as the result in Step 4). If the invocation is successful the resulting response is first prepared (again, possibly requiring modification to the message) before being returned to the Service Consumer (Step 4) and concluding the invocation process.

6.4 Demand-driven Dynamic Deployment

In a dynamic, demand-driven system, endpoint deployment may occur in response to an existing or anticipated event – in this case, the invocation of an operation of a Web Service for which there are currently zero active endpoints. The process presented below will be familiar – it is a composition of two processes presented previously: Web Service endpoint deployment and the invocation of a deployed endpoint's operations.

When the ActiveServiceDirectory receives a request for all endpoints of the 'DayTime' Web Service, it finds there are none currently deployed. In this case it is up to the ActiveServiceDirectory to initiate (though not enact) the deployment of the first instance of the requested Web Service by contacting that Web Service's Manager and invoking the 'requestFirstInstance' operation. At this point the deployment process is executed identically as previously presented; when the Manager returns from the 'requestFirstInstance' operation the ActiveServiceDirectory can re-perform the lookup and return either a reference to the newly deployed endpoint, or a meaningful error indicating that, given the currently available resources in the infrastructure, it is not possible to fulfill the request at the given time.

The process represented in Fig. 25 below details the steps from the initial Service Consumer request through the demand-driven dynamic deployment of a 'DayTime' endpoint, its subsequent registration in the ActiveServiceDirectory, the return of a list of active 'DayTime' endpoints to the local point of presence, the performance of the original invocation request on the newly deployed endpoint and, finally, the return of results to the Service Consumer.

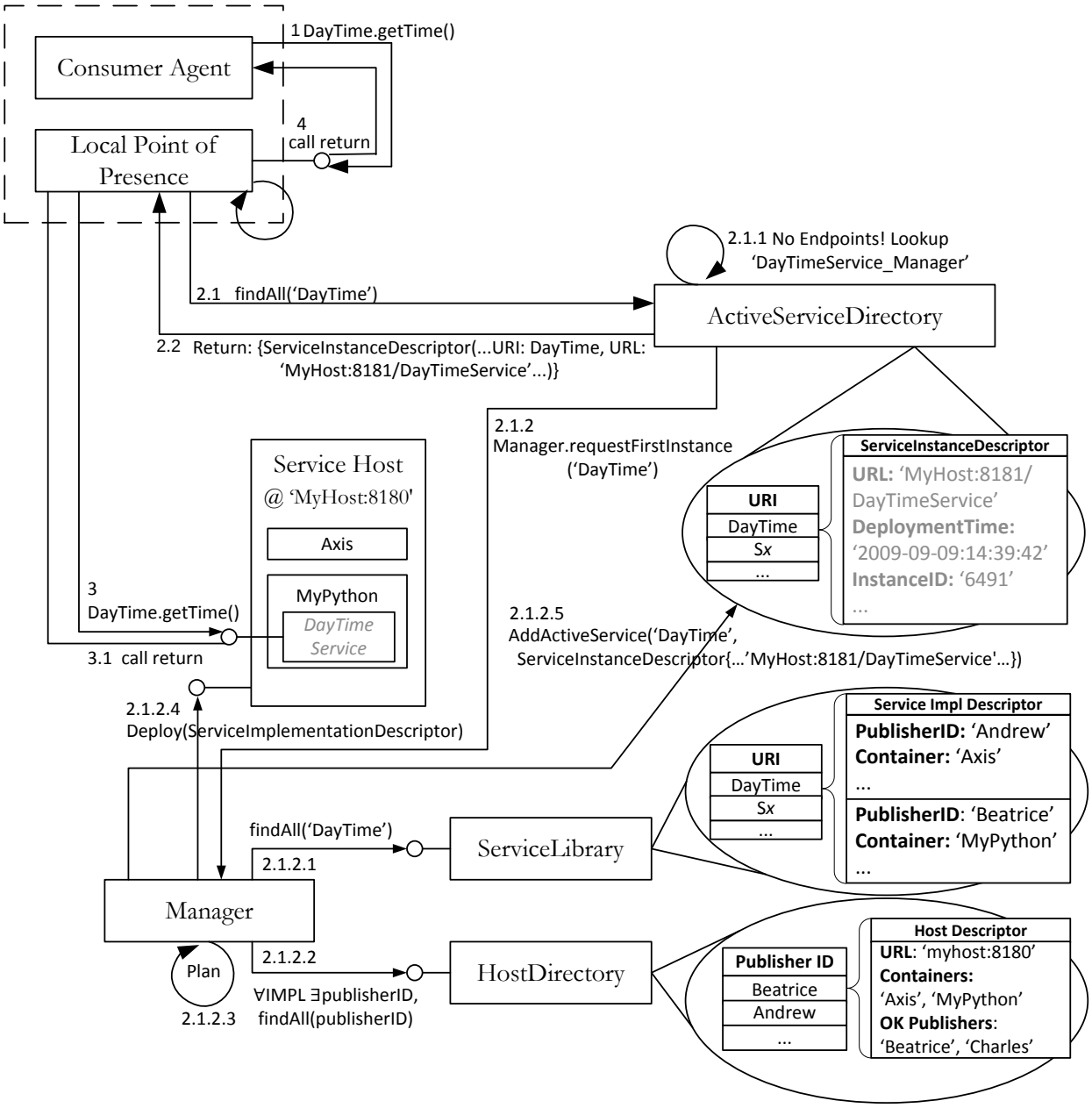

Fig. 25 Demand-driven dynamic deployment of the first 'DayTime' Web Service endpoint. Deployment is carried out in response to demand for a service which initially has no deployed endpoints.

It should be mentioned again at this point that the Manager for a Web Service does not need to be currently deployed in order for that Web Service to be deployable. If implemented as a Web Service – and with generic and/or custom, service-specific Manager implementations published in the ServiceLibrary – the exact same deployment mechanisms can be used to deploy and manage a Web Service's Manager as those Managers use to manage their particular Web Service. This recursive, collapsible model enables an infrastructure to be 'wound up' to provide enough Web Service endpoints to meet demand, and then 'wound down' to a state of

zero resource consumption when demand falls to zero. (Note that the infrastructure must start somewhere and, even in a period of zero demand, a passive boostrapping process [or similar mechanism] capable of deploying the first 'ManagerManager' is necessary; resource consumption can thus never be said to be 'completely' zero).

In pursuit of a design where an idle infrastructure consumes low levels of hosting resources, the described mechanisms and framework for the publication, invocation, deployment and management of Web Services can also be used for the provision of the infrastructure components themselves. Detailed in the next chapter, entitled "Reference Implementation", the ActiveServiceDirectory, HostDirectory, and ServiceLibrary are all published as Web Services, each deployed and managed by a generic Manager capable of detecting and reacting to changes in consumer demand, deploying and undeploying endpoints as necessary to balance resource consumption while providing Service Consumers with highly available Web Services. The properties provided to Service Consumer (reliability, robust invocation techniques, highly available Web Services) themselves become properties of the architecture itself – a reflective design that both simplifies and improves the quality of interactions for all participants in the Web Service lifecycle.