Considerations for Versioning SOA Resources

Kenneth Laskey[#]
The MITRE Corporation
klaskey@mitre.org

Abstract

Service oriented architecture is a paradigm for bringing together needs and capabilities, where SOA services provide an effective means of connecting consumers and the means to realize desired real world effects. The resources accessed as part of SOA interactions are independently owned and evolved but must be unambiguously identifiable. In cases where the resources are changing, the consumer must be able to evaluate how those changes affect appropriateness for use, whether those are changes to the underlying capabilities, the service access, or the service description. This paper presents early discussions on versioning in the context of a SOA reference architecture.

1. Introduction

The OASIS SOA Reference Model Technical Committee (TC) has produced the Reference Model for Service Oriented Architecture (SOA-RM) as an abstract framework for understanding significant entities and relationships among those entities within a service-oriented environment [1]. The TC has continued its work by developing a SOA Reference Architecture (SOA-RA) to provide an abstract realization of SOA, focusing on the elements and their relationships needed to enable SOA-based systems to be used, realized and owned; while avoiding reliance on specific concrete technologies [2].

As part of its work, the TC is considering the principles and architectural implications of versioning of SOA resources, in particular the versioning of SOA services. While the necessity of versioning mechanisms may be obvious to anyone who has dealt with SOA services, the topic rarely penetrates the mainstream of SOA publications and implementations [3]. The work presented here captures important considerations and begins to lay the framework for the modeling to be incorporated into SOA-RA. Still in its early stages, the ideas provide the basis for discussion and further contributions by the wider community.

[#] The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author.
Based on "Considerations for SOA Versioning", by Kenneth Laskey which appeared in *Proceedings of the Enterprise Distributed Object Computing Conference Workshops, 2008. EDOCW '08.* © 2008 IEEE.

2. General discussion of versioning

Versioning assumes simultaneous existence of multiple (different) implementations of a given resource, with every implementation distinguishable and individually addressable [3]. For the current discussion, we define versioning as the process of systematically cataloguing the changes to a resource. This implies an identifiable resource, a specific set of revisions to that resource, and a modified resource that is the result of applying the revisions to the original.

A version identifier is a unique label that indicates a specific configuration of a resource. For software systems, it is often composed of an immutable name (e.g. example.txt) and a varying string of nonnegative integers separated by decimal points (e.g. 3.2.1). While this is a commonly observed scheme, it is by no means universally used or used consistently. For example, Microsoft's major release versioning of its operating system is Windows 95, Windows 2000, Windows XP, Windows Vista. For Apple, the current major versioning is 10.1, 10.2, 10.3, 10.4, 10.5 but this is not consistent with what version numbers meant prior to OS X. Other variations are described in [4].

From these examples, we can conclude that not only should a version be specified through use of a version identifier, but an explanation should be available on how the identifier is to be interpreted. For the current discussion, we consider the format of this explanation to be out of scope, but it should be readily accessible and understandable by those needing to interpret the versioning and consistently applied to enable appropriate use of the versioned resource.

Note, the current discussion does not preclude a service lineage where there may be branching of one version giving rise to more than one subsequent version, or two or more versions being merged into one. However, if the descendants are considered to be versions of the original instead of a new entity, then the explanation of the versioning scheme must clearly explain how the version identifier is to be interpreted. For simplicity, this paper will not specifically deal with such cases because these do not significantly impact the issues being raised or the associated conclusions.

3. Versioning and compatibility

Beyond a version identifier indicating a consistent sequence of versions and defining the revisions that transform one version into the next, it is necessary to define a use strategy to specify how a command set or information set designed for a predecessor is to be used by the current version and, conversely, how a command set or information set for the current version is to be processed by previous versions.

We also refer to a consumer of the resource, where the consumer is prepared to engage a certain version of the resource and where the concepts of compatibility and sufficiency depend on how well a consumer accustomed to one version can deal with a predecessor or future version of the resource.

For brevity, we will not discuss notions of compatibility here – an elementary discussion can be found in Wikipedia [5][6] and a more technical consideration is documented by the W3C Technical Architecture Group [7][8]. However, it should be noted that compatibility is determined with respect to a revision and the resource or consumer reflecting that revision. Compatibility depends on context. In particular, a general statement that something is backward or forward compatible is meaningless unless it is stated against what is compatibility being assessed. In addition, something can be designed with forward compatibility in mind, but it cannot be deemed forward compatible until it can be exercised against a specific future revision.

4. Versioning and sufficiency

We consider compatibility between versions because it is highly desirable to maintain reliable communications and to realize some aspects of the desired results when changes occur for a resource. As such, an older resource can receive a message constructed for a newer version, can process the message in terms of its understanding, and will perform functions consistent with its older context. Similarly, an older consumer may receive a response that contains unexpected information and may just make use of the content it finds consistent with its older context.

The question not addressed by compatibility is whether the results of interactions across versions is sufficient for the intent of the consumer. A new resource may have new functionality that can be invoked through new terms in the schema used by the message payload. The new terms could be added through an extensibility mechanism built into the original schema, and thus the original schema may be able to validate the new payload without understanding that new functionality is desired. The target resource can generate and return reasonable results but not necessarily the results required by the consumer. Other aspects of versioning policy and the need to consider consumer business needs when assessing compatibility are discussed in [9].

Thus, we see where it is not only necessary to have a versioning strategy that defines the semantics of the version identifier used by any resource, but it may also be necessary to have versioning policies on the part of the consumer that define what compatibility approach is appropriate when interacting across versions. Note that while policies for SOA are typically addressed in terms of the service policies, the consumer may also have policies, and the two policy sets must be reconciled if interaction is to proceed.

A versioning scheme for a service may include a generic policy, such as any succeeding version identified as 1.j will be backward compatible with any 1.i previous version in the sense that results are identically generated in version 1.j for functionality that existed in previous 1.i versions. (Note here we assume $i<j$ implies a previous version.) In this case, an adequate policy for the resource may be that any 1.j compatible request can be processed by any 1.i version of the resource; an adequate policy for the consumer could be that a response from any 1.i version of the resource is acceptable.

If version 1.j functionality is required, then while there exists a degree of compatibility in understanding and processing version 1.j requests, only version 1.j or later is sufficient for the consumer's needs, and use of 1.i versions is unacceptable.

The issue can be sidestepped if every version of the resource is reachable through a different endpoint and the consumer explicitly chooses the endpoint and thus the version to be used, but a resource provider may simply want to reuse the endpoint for new versions. For example, there is no externally available version for Google but its ranking algorithms are often altered to increase search fidelity or to simply respond to efforts by content providers to game Google's ranking algorithm. Google has had a single, stable endpoint at www.google.com, and consumers use whatever version is currently accessible from that endpoint.

In general, a search engine user expects results will be different for a search done last week and the identical search repeated this week. Items that became known to the search engine in the past week would now be presented in a consistent manner. However, the user has also learned to expect (or simply ignores) that the search engine itself may have been changed and the results could vary even if there were no additional items added to the engine's index. The unstated context becomes the default versioning policy.

5. Versioning for SOA services

Versioning indicates change and the question then is what changes are necessary to be reflected in a SOA ecosystem.

The OASIS SOA Reference Model (SOA-RM) identifies the dynamic aspects of SOA services as visibility, interaction, and real world effects. We will use this as the framework in which to investigate versioning for SOA.

Additionally, in a manner consistent with SOA-RM, we assume service access to any underlying capability; while this is not necessary, it does simplify the discussion. However, the discussion is generally applicable to any resource, whether or not it is accessible through a SOA service.

In the SOA Reference Architecture (SOA-RA), the model for a resource (Figure 1) states that resources have descriptions and the descriptions reference one or more identifiers by which the identity of the resource is established. SOA-RA goes on to state that both services AND service descriptions are resources. While this may seem circular in reasoning, SOA-RA models the general concept of description and then expands on the model for service description as an extension of the general description model. The rationale for this should become clear below.

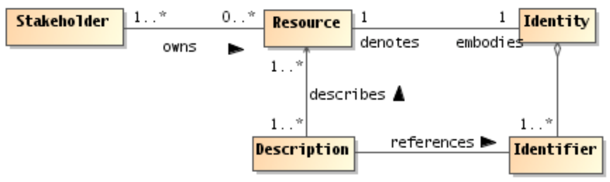

Figure 1 SOA-RA Resource Model

A service description provides information on

- what a service does, i.e. its business functions, the specific real world effects resulting from actions against a service, and technical assumptions that constrain the applicability of the results;
- how to communicate with the service, i.e. the semantics and structure of a message payload sent to the service and the actions that sending such messages can invoke;
- conditions for using the service, i.e. applicable policies;
- metrics indicating service performance; and
- details of reaching the service, i.e. the service endpoints and protocols to be used at those endpoints.

Thus as reflected in description, changes could affect the functions a service provides, the mechanics for interacting with it, the conditions for interacting with it, or knowledge of how the service will respond. Any change could derive from the underlying capability or the service as access to that capability. However, the SOA principle of opacity says the consumer cares only about what results from the interaction, so the specifics of where the change occurred in the implementation is generally irrelevant.

From this, we conclude there is a need to version the service as part of its service description, but it is not necessary to explicitly capture version information about component capabilities or component services from which the service of interest is constructed. These components likely have their own configuration management and versioning conventions, but these are generally not of interest to the service consumer. Considerations related to this will be discussed below.

6. Versioning of service description

In the previous section, we discussed how a service could change and how this would be reflected in its description, but there are possible changes in the description that may not directly derive from changes to the service. Consider, a service description could change to reflect:

- correcting errors that do not significantly change the description, e.g. a simple typo;
- correcting errors that do significantly change description, e.g. the word NOT was missing from the functionality description;
- adding information, e.g. an additional real world effect that was previously considered inconsequential;
- removing information that was previously required or thought useful, e.g. the number of times the service has been used;
- consolidating elsewhere the specifics of some information and replacing the occurrences in the service description by a link to the consolidated location, e.g. version history.

The degree to which these changes are important will likely depend on the context of use. Fixing the typo should be innocuous, but there may be occasions where it will affect someone's interpretation of surrounding information. The second and third items can be seen as potentially substantive changes, and the last two items may affect the ease with which a consumer can process descriptive information. Thus, any change in the service description should be reflected in a new version for the description, where the version identifier may be constructed to indicate the expected significance of the change.

It should be noted that a principle from SOA-RM is that description is inherently incomplete – one can never describe every aspect of a resource. It is also possible that different aspects of description will be captured by different description sets. For example, the configuration management for a service will likely contain implementation details that are not consistent with the service description to support SOA interaction. In the current discussion, we only consider the description needed to enable and support service interaction. References in the following to one description do not preclude the existence of complementary descriptions expressing other aspects of service description, but a full discussion of such descriptions is beyond the scope of this paper.

Having established a focus on service interaction, we consider how information in the service description affects multiple versions of the description. As an obvious first requirement, the description should unambiguously identify its subject resource; if the resource is versioned, the identity of the resource should indicate its version and the explanation of the resource versioning scheme. (Note, the explanation may be indicated by a link to external documentation.) As a consequence of this requirement, each version of the service should have a unique description, i.e. each new resource version has a corresponding new description, even if the only thing to change is the identifier indicating resource version. For example, if an error was found in the resource implementation and no new functionality or conditions of use were introduced when the error was corrected, the previous description would otherwise still be valid, but a new description would be needed to identify the corrected service version.

What then of versions of the service description for the same service version? As indicated above, a new version of the service description would be required if the description changed but the service did not. However, given the description is of a single version of a resource, a new version of the description would supersede any previous one for that resource because the description update would take precedence. Distinct versioning of the description does allow review of past descriptions, for example if a decision to use or not use a service has changed as a result of the change to description.

7. Possible representation of service and description versions

The resource model in Figure 1 shows the relationship between a resource, its description, and its identifier. In this section, we will consider a possible representation for these that can support the agility desired of SOA and the clarity to unambiguously identify and describe the resource. Recall that the description is also considered a resource and while much of the following discussion will focus on a SOA service, any general points on identifying a resource will also apply to the service description.

It should be noted that the following are the starting elements of ideas on how identity and versioning can be approached and do not imply consensus on the approach as presented.

The basis of the current discussion is to use the Uniform Resource Identifier (URI) [10] to identify resources. The URI has shown its ability to create a uniform address space for the Web, and many are of the opinion that this is one of the prime enablers for the Web's success.

Let us assume *service1* is a resource that may have numerous versions, and we identify a particular version by the URI *http:///a.b.c/services1/20080601/*. We define the rightmost field as a date of the form *yyyymmdd* and the preceding field to be the resource name. The name service1 is arbitrary for this discussion and may be replaced by any legal URI path; the authority following // can likewise be any legal string for this portion of the URI. For example, if the service is identified by *http://a.b.c/services/preferred/service1/20080601/*, the resource name is still service1 and the version corresponds to the date 20080601. Application specific guidance can be provided for generating the resource name or other parts of the URI.

The date acts as the discriminating identifier for this version of service1. Date is used instead of a version number because version numbering schemes can change over time and having the version number as part of the identifier could lead to eventual confusion and inconsistencies. The date scheme is more stable and universal. As the identifier, the date would represent some definable milestone in the resource life cycle, and the versioning scheme could define exactly what life cycle stage was being referenced.

One perceived drawback in using the date as part of the identifier is that, from a business perspective, one may not want to associate a date from two years ago with their service because the service may then be perceived as old technology, and competitive services may gain an advantage by implying newer technology. Further discussion will be needed to determine if this is a critical issue.

Given the URI as presented as the service identifier, we propose that the result of dereferencing this URI, i.e. typing it into a browser, will return the latest service description. This removes the ambiguity of where to find the description of the resource and tightly binds the resource to the description. This can be especially important when there are multiple versions of the service.

How can this work across service versions? Let us refer to the collection of service versions as the *service family*. If each service version is identified by a unique URI as defined above, let us require that every service family host a standard file – let us say, for example, version-identifiers.html – that when dereferencing would return a list of all identifiers that comprise the service family. If we assume persistence of the descriptions, these can be accessed and examined even if the described services are no longer available.

Note, we have emphasized the link between the identifier and the description but we have not discussed the service version or the endpoint where the service would be accessed. Every service description should identify the version of the service it is describing and the definition of the versioning scheme being used. For example, a service description could contain

```
 <version source="http://a.b.c/versiondef/20080215/">
     6.7.2
 </version>
```

to identify the version number to be 6.7.2 as defined by the versioning scheme identified by (and possibly retrievable from) http://a.b.c/versiondef/20080215/. The service description would also point to the corresponding service endpoints, either explicitly or through reference to the service WSDL.

Identifying versions of service description could follow a similar pattern. For the service identified by http:///a.b.c/services1/20080601/, dereferencing the URI would return the latest service description. To identify the versions of service description corresponding to this service, let us create a description identifier by appending the date a particular service description becomes active to give, for example, http:///a.b.c/services1/20080601/20090102. Further, let us assume every service hosts a standard file – let us say, for example, description-identifiers.html – that when dereferencing would return a list of all identifiers that comprise the descriptions associated with the service. Again, for archival purposes, this would enable access to older versions of description.

8. The challenges of service opacity

A guiding principle of service oriented architecture is that the consumer should be able to use a service without concern or interest in the implementation details. We typically focus on the WSDL (Web Services Description Language) representation of the service interface: the abstract specifics of the exchanged information and the implementation details of how and where the exchange occurs. However, while a stable interface is of obvious importance in providing "loose coupling" for consumers, it is naive to expect that implementation changes are of no interest. For example, if a service accesses a new data source to respond to a query, the consumer would want to be aware of this if returned values start showing a different pattern from past experience. This is especially true if the new pattern emerged without any obvious action on the part of the consumer.

The question remains how to balance the value of opacity of implementation against the real need of being able to consider the implications of underlying change. The answer is likely tied to the description of the service configuration. While the details are probably separate from the service description supporting interaction, the service description may reflect such changes at a macro level, such as the version of the configuration that would in turn indicate the separate versions of components comprising the service. If there was need, the identified configuration and previous configurations could be accessed to investigate what changes have occurred and to decide compatibility in the context of such changes.

9. Conclusions

A thorough understanding of what versioning means in the context of service oriented architecture is one of the current topics of discussion for the OASIS SOA-RA subcommittee developing a SOA reference architecture. The analysis to this point has indicated the importance of defining and applying a well-documented versioning strategy for resources such as a SOA service. The discussion has also touched on a related versioning strategy for the corresponding service description. From the perspective of the service owner/provider, the description should unambiguously identify the service, the business functions it provides and the results it generates, the means to communicate with the service, the means to access the service, the conditions of use, and metrics on service operational characteristics. A change in version should reflect a change in any of these aspects of description. While the service description would not provide details of service implementation that should be opaque to the consumer, the description should indicate when such changes occur so the consumer can assess changes in using a service over time.

The work presented here is preliminary and likely to evolve with continued discussion. The public review draft of SOA-RA, available from [2], discusses the SOA ecosystem in its support of business activities, an expanded discussion of service visibility and interaction, and aspects of owning SOA resources, such as SOA governance and security. An updated public review draft is due out shortly, and the SOA-RA subcommittee looks forward to comments and suggestions.

10. References

[1] "Reference Model for Service Oriented Architecture", C.W. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz eds, OASIS Standard, 12 October 2006, http://www.oasis-open.org/specs/index.php#soa-rmv1.0.

[2] "Reference Architecture for Service Oriented Architecture", J. A. Estefan, K. Laskey, F. G. McCabe, D. Thorton eds., Public Review Draft 1, 23 April 2008, http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.html.

[3] B. Lublinsky, "Versioning in SOA", Microsoft Architect Journal, April 2007, http://msdn.microsoft.com/en-us/library/bb491124.aspx.

[4] Software versioning, Wikipedia, http://en.wikipedia.org/wiki/Versioning.

[5] Backward compatibility, Wikipedia, http://en.wikipedia.org/wiki/Backward_compatibility.

[6] Forward compatibility, Wikipedia, http://en.wikipedia.org/wiki/Forward_compatibility.

[7] D. Orchard, "Extending and Versioning Languages: Terminology", Draft TAG Finding, W3C Technical Architecture Group, 13 November 2007, http://www.w3.org/2001/tag/doc/versioning.

[8] D. Orchard, "Extending and Versioning Languages: Strategies", Draft TAG Finding, W3C Technical Architecture Group, 13 November 2007, http://www.w3.org/2001/tag/doc/versioning-strategies.

[9] M. Poulin, "Service Versioning for SOA", SOA World Magazine, 26 July 2006. Available at http://soa.sys-con.com/node/250503.

[10] Uniform Resource Identifiers (URI): Generic Syntax, IEFT RFC 2396, August 1998. Available at http://www.ietf.org/rfc/rfc2396.txt.