Extending WSDL with versioning information

Henry Been
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
h.been@student.utwente.nl

ABSTRACT
More and more businesses are providing web services to interact with their business applications, allowing for loosely-coupled software solutions, possibly spanning multiple businesses. Demand for more functionality forces web service providers to host multiple versions of the same services. Some consumers might upgrade to the newest version, but some might not. This increases the number of running versions and the time needed to maintain them. A survey is done of requirements posed on web service versioning and current approaches. This paper proposes a lightweight and visible approach to enrich WSDL files with versioning information. This allows service providers to inform service consumers about changes and newer versions and encouraging consumers to upgrade quickly to the new version.

Keywords
WSDL, web service versioning, web service evolution, metadata

1. INTRODUCTION
More and more businesses are providing web interfaces to interact with their software, allowing other businesses to integrate their IT solutions with each other. The interfaces of web services change over time, just as the interfaces of traditional software like shared libraries and new versions become available. When using traditional software a consumer can decide for himself which version of a library to use, so he can keep using an older version even when it is no longer supported. With a web service this is not possible, since it is the service provider who decides on which versions will be made available.

If the provider publishes a new version of an object and its WSDL at the same URL as the previous version this might, and often will, break consumers of the service. To avoid this, the current best practice for dealing with multiple versions of a web service is keeping them available in parallel [1, 3]. Each new version of a web service is published at a new URL and is described in a new WSDL file. This does not break any existing consumer, but is not visible to the client of the web service either. For the client there is not only no need to upgrade to a new version of the web service, but the client stays unaware of the existence of a new version. One solution extending this approach is developing a chain of adapters [6] implementing previous versions with a small service that calls the next version of the web service. When introducing a new version only the former version has to be replaced by an adapter, extending the chain with one version. Part of the process of developing adapters can already be automated by tools [10], reducing the burden on the web service developer. This solution is fully backwards compatible, does not introduce any duplicate code and is easy to use for both the web service developer as consumer.

The main drawback of this approach is that it does not allow for informing the web service consumer of the fact that he is using an older version of the web service. When more new versions are introduced, the set of available versions increases and the chain of adapters grows larger unless older versions can be taken off-line. Since users might keep using older versions, either because they don’t know or don’t care about the new version, older versions keep being used. The web service provider has to choose between taking off-line older versions and force the consumer to migrate or taking the penalties for this long line of adapters.

1.1 Research questions
The problem that this paper aims to solve is the lack of versioning information that is available in a WSDL description of a service. To do this, the following questions have to be answered:

1. Which requirements on versioning web services can be drawn up?

2. Which approaches to supporting web service versioning do currently exist?

3. How can a WSDL extension be designed that fulfills the requirements posed on versioning web services?

To answer the first question a literature study has been undertaken to discover current mechanisms used to enrich software with version information. This study is not limited to approaches currently in use with regards to web services, but is extended to the domain of traditional software. The resulting requirements are described in section 2. The second research question will be answered in section 3 where current solutions and best practices together with their disadvantages are discussed. Section 4 delivers the proposed approach, which will be evaluated in section 5.

1.2 Terminology
Whenever the term service is used in this paper, unless stated otherwise, it is assumed to be a web service. When
the terms provider and consumer are used, they refer to either the software package that provides or uses the service or the developers responsible for maintaining them. Only when needed a distinction will be made between the consumer and the consumer developer and provider and provider developer.

2. REQUIREMENTS
In this section a number of requirements on service versioning are described. To illustrate the requirements in this section, a sample service for accessing information on the weather is assumed. A number of versions of this service are shown in figure 1 as a graph. Listing 1 shows the source code of the objects that are exposed as the versions 1.0, 1.1 and 1.2 of the weather service.

![Figure 1. A number of versions of our weather service.](image)

```java
public class Weather10 {
    public int getTemperature() {
        ...
    }
}
```

```java
public class Weather11 {
    public double getTemperatureAtDateTime(Date moment) {
        ...
    }
    @Deprecated
    public int getTemperature() {
        ...
    }
}
```

```java
public class Weather12 {
    @Deprecated
    public double getTemperatureAtDateTime(Date moment) {
        ...
    }
    public double getLocalTemperature(Date moment, String location) {
        ...
    }
}
```

Listing 1. Versions 1.0, 1.1 and 1.2 of the weather service

2.2 Visibility
The consumer should become aware of the existence of different versions of the service and even be encouraged to upgrade to a newer version

When providing web services there are two opposing forces. On one side there is the demand for change and improvement of web services leading to new versions of a service, but on the other hand there are existing users who want to keep using older versions of the service. To satisfy the demands made by both sides, often multiple versions of a web service are available at the same time. For the service provider this creates the overhead of managing more web services, especially when new versions follow each other quickly. This is why an important requirement on any mechanism that deals with service versioning is, that it should do so in a way that is visible to the consumer. The consumer should be aware of the fact that there are multiple versions of the service running and how they relate to each other. This visibility of the different versions to the consumer will increase the chance that he will upgrade to a newer version of the service. If more consumers upgrade faster to newer versions, the producer has to maintain less versions in parallel. This reduces the burden of maintenance and allows for the quicker introduction of new versions.

2.3 Valid WSDL
Any WSDL file with version information added should still be valid according to the WSDL specification

Any client retrieving a WSDL file with version information added to it should be able to parse and validate it according to the WSDL specification [2]. This limits any approach to use the extensibility of the WSDL language by introducing extra tags as part of the operations, interfaces, bindings and endpoints definitions in the WSDL file. These new tags should belong to another namespace than the tags that are part of the WSDL. Introducing information in any other way might break parsing of the WSDL description by clients that are not aware of the extension for versioning services. By using the WSDL extensibility extra information will be ignored by unaware service consumers, while the information is still there. This allows for providing the same WSDL file to any consumer and allowing those that do support the extension to pick up the extra information. As long as there is no tool support for the versioning mechanism this can even be done by hand by the developers of the service consumer.

2.4 Deprecation and guaranteeing of operations
When adding versioning information to a service the developer should be able to encourage or discourage the usage of specific operations

Deprecation of operations is a well-known mechanism to inform the user that the use of a specific operation is discouraged. Marking an operation as deprecated informs the user that the marked operation might be removed in a next version of the service. But this is only part of the story; the service developer might also want to tell why the operation is deprecated and which alternatives are available to assist and encourage the service consumer to stop using the deprecated operation. Any proposed version solution should support these three elements of deprecation.

As opposed to deprecating operations, the service provider might want to encourage the use of specific operations which are guaranteed to still be present in newer versions of the service. The need for making the distinction be-
between not only deprecated and non-deprecated operations but also encouraged methods resides in the fact that new versions of a web service can quickly follow each other and the service provider can deliver guarantees to the service consumer on which operations to stay and for how long they are guaranteed.

2.5 Minimal change after introduction

The number of times a versioned WSDL description should be updated should be once at maximum.

After introducing a new version of a service and its accompanying WSDL file there should be ideally no need for updating the WSDL description. Unfortunately the provider developer cannot predict what will change in future versions when introducing a new version. This implies there will always be the need to update the description of a versioned WSDL after the introduction of new versions. For this reason any versioning mechanism should be limited to describing only the difference between two versions that are directly connected in the version graph. In this way the consumer gets retrieve information relevant to successor versions out of a WSDL description. He can use this information to either upgrade to the next version or traverse this linked list of descriptions until he arrives at the newest version.

3. EXISTING SOLUTIONS

Parachuri and Mallick [8] describe a number of different service versioning approaches. The most primitive form they see is document style versioning where version information is stored within the document and no tools are used to automatically manage the versions of the service. It can be used when the number of services to govern is small.

Another simple approach is the use of XML versioning [3]. XML versioning can be applied by either including the version identifier in the targetName space of the WSDL description and update it accordingly, or setting a version identifier on the XML document as a whole.

An more heavy-weight approach is to make the service registry responsible for maintaining links between versions. A completely different approach is that of WSDL versioning.

3.1 Registry based

Fang et al. [4] introduce a service registry that supports the multiple versions of the same service. Providers can publish a new version of a service and relate it to an existing version. The description of the services themselves, in WSDL, does not contain any versioning information, since this is provided to the registry when registering a new service. In this approach the registry is responsible for maintaining the relations between different versions and providing consumers with a notification of a new version whenever necessary.

This approach is extended by Leitner, Michlmayr, Rosenberg and Dustdar [7] who introduce dynamically binding clients. When a new version of the same service is registered in a registry, the registry scans the accompanying WSDL file for changes since the former version. Based on this scan the new version is classified as either backwards compatible or not. The consumer proxy using the service can periodically check the register for new versions of the server that are backwards compatible and bind to this new version automatically.

This approach tries to lighten the burden on the consumer when working with changing services by automatically upgrading to newer, compatible versions, but does not allow for providing any versioning information to the consumer developer about this. When dynamically rebinding, the consumer might get stuck on a version whenever the next version is no longer backwards compatible. Since the consumer will never get notified about this fact the versioning approach is not visible. Because the existing WSDL descriptions are only analyzed and not enriched, there is no way of finding out about different versions without using a registry and the provider cannot deprecate and guarantee specific operations. An advantage of this approach is that a WSDL description does not even has to change once after its introduction.

3.2 WSDL versioning

A different approach that directly addresses the impossibility of adding version information to WSDL files is proposed by Juric, Sasa, Brumen and Rozman [5]. They propose to integrate the descriptions of different versions of the service into one master description of all versions. This approach provides support for managing multiple versions in parallel by defining the different versions, describing only the differences between a sequence of versions and the deprecation of operations or complete versions. The drawback of this approach is that this master description cannot be used directly by service consumers that are unaware of this versioning approach. The service provider has to adapt the server to support these version-unaware clients by accepting requests for WSDL files, retrieving the default version form the master description and returning it as a normal WSDL file. Since this extra server has to be deployed and maintained the burden on the service provider is increased.

This approach also violates the requirement of visibility. The way versioning is performed is hidden from any consumer that is not aware of this approach. These consumers have to rely on an adapted server to provide him with unversioned WSDL descriptions. If the developer of a service consumer inspects the WSDL description it does not contain any information related to the version of the web service, since this information has been lost due to the translation made from the master file to a standard WSDL description. The requirement of full backwards compatibility is also broken since version-unaware clients are always provided with the default version of the service as defined in the master WSDL file. When this default version changes, any version-unaware client might still be broken since they are suddenly confronted with another version of the service.

4. PROPOSED APPROACH

This paper introduces a set of small and lightweight extensions meant to be inserted into existing WSDL documents, annotating them with versioning information. The extensions follow the rules for extending WSDL [2] to guarantee that any service consumer unaware of the extensions can still parse, validate and use the resulting extended WSDL files. A new namespace ‘http://utwente.nl/wsdl’ has been used to identify the tags part of the extension and insert them as child tags of the information elements that are part of the WSDL.

Our approach shows similarities to the approach chosen by Juric, Sasa, Brumen and Rozman [5]. The main difference is that our approach aims at delivering a lightweight, easy to use extension that can be used without any specific tooling, servers, proxies or frameworks. While this extension was designed before the approach delivered by Juric, Sasa, Brumen en Rozman was discovered, a number of principles are applied in the same way. Whereas Juric
et al. gather all WSDL descriptions and bundle them into one master file, our approach aims at inserting any version information inside WSDL files. Our approach keeps the original structure of the WSDL files intact and only adds new tags to the bottom of the XML tree. This is done to comply with the requirement of maintaining valid WSDL files for version-unaware clients without introducing the need for an extra or adapted server to serve version unaware clients.

4.1 Versioning
To successfully implement any versioning mechanism one has to start with introducing a way to identify different versions of the service. This is done by adding a version tag as a child to any interface tag in the WSDL file. The version tags consist of two mandatory attributes name and lastVersion. The name attribute can be freely chosen by the service developer on the condition that the name is unique for all versions of the interface. The lastVersion attribute is a boolean to indicate whether this is the latest version of the interface or not. An optional attribute discontinuedAt is available to inform the user that the service will be taken off-line at the specified date.

The nextVersion tag is used to indicate that a new version of the service has been introduced that succeeds the current version. The nextVersion tag has two mandatory attributes. The name attribute is used to identify the new version and the location attribute is used to locate the new version.

The versionInfo tag is introduced to further describe the version. This optional tag can be used to give a textual description of the version and can be used by the service developer to briefly describe this version and what is different from previous versions. This tag is introduced by Juric et al. and fits in the proposed approach [5]. The version, versionInfo and nextVersion tags are illustrated in listing 2.

```xml
<w:description targetNamespace="http://www.w3.org/ns/wsdl">
  <w:interface name="Service Interface">
    <w:version name="1.0" lastVersion="false" discontinuedAt="2011-01-01"/>
    <w:versionInfo>
      <w:nextVersion name="1.1" location="http://localhost/weather/1/1"/>
      <w:deprecation date="2010-12-02"
        reason="Replaced with getTemperatureAtDateTime"/>
    </w:versionInfo>
    <w:getLocalTemperature style="http://www.w3.org/ns/wsdl"/>
    <w:return type="http://www.w3.org/ns/wsdl/
      in,return">
      <w:signature name="getTemperature"/>
    </w:return>
    <w:operation name="getTemperature"/>
    <w:port name="getTemperaturePort">
      <w:operation name="getTemperature"/>
    </w:port>
  </w:interface>
</w:description>
```

Listing 2. Partial versioned WSDL of version 1.0 of the weather service after the introduction of version 1.1

4.2 Deprecated
To support deprecation of operations the deprecation tag is introduced. This tag can be used to hint that usage of a the deprecated method is strongly discouraged. The tag has two optional attributes. The first attribute deprecationDate can be used to identify when the method was deprecated. The second attribute removedIn can be used to identify one or more versions of the service where the deprecated operation will no longer be available. The service provider can also add an extra child tag reason to elaborate on the reason for deprecating the method. An example of the deprecation tag is given in listing 2.

```xml
<w:deprecation date="2011-12-02" reason="Replaced with getTemperatureAtDateTime"/>
```

Listing 3. Partial versioned WSDL of version 1.2 of the weather service after its introduction

4.3 GuaranteedUntil
Oposed to discouraging the use of operations, the service developer might feel the need to encourage the use of a specific operation. He can do this by guaranteeing the availability of the operation for a certain time. This is done by introducing the guaranteed tag with the mandatory attribute expireDate. An example of this tag is delivered in Listing 3.

```xml
<w:guaranteed expireDate="2011-12-02"/>
```

Listing 3. Partial versioned WSDL of version 1.2 of the weather service after its introduction

4.4 Enriching WSDL files
The proposed approach is based on adding versioning information to existing WSDL files. There are three moments in the life cycle of a service when this should be done. The first is when introducing a new version of the service and its accompanying WSDL file. In this file the service developer should at least name the new version by adding a version tag and setting the lastVersion flag. Of course the service developer is encouraged to add more information regarding the deprecation and guaranteeing of operations, but this is not mandatory. The second moment in a versions life when its WSDL file should be updated is when introducing a new version that is intended to replace it. When this happens the lastVersion flag should be set to true and a nextVersion tag should be added to relate the old and new version. Finally a service developer might choose to indicate that a specific version might be taken off-line at some point in time. He can then update the WSDL file for the third time to give a strong signal to any final consumers to stop using this version and upgrade to a newer version.

By building a linked list of versions and relating versions only to those directly succeeding it the number of times a WSDL file might have to be updated is limited to a maximum of three occasions: The introduction of the version, the introduction of a succeeding version and before taking the version off-line. This is done intentionally to reduce the burden on the developer who only needs to think about the version he is currently launching and the versions it
will replace.

5. DISCUSSION
The most important requirement, is the requirement of valid WSDL as stated in section 2. The validity of WSDL files that are enriched with the proposed extensions has been established by using them with the PHP Soapclient [11] as well as Axis2 [9] (as an Eclipse plugin). Both tools are still able to use the enriched files in exactly the same way as the original files. By choosing an approach where information is added in-line at the appropriate place the proposed extension aims at maximum visibility to the consumer. The tags documenting a version, next versions, deprecated and guaranteed operations are all designed to be a part of the tag they describe. Finally all the tags are designed in such a way that they can be added to the WSDL description when the version is introduced or when a new successor is introduced. In the running example this means that the WSDL description of version 1.0 only has to be enriched at its introduction, at the introduction of version 1.1 and the introduction of version 2.0.

Possible improvements
The extensions proposed in section 4 are aimed at static clients like the PHP Soapclient [11] who obtain the WSDL file accompanying a web service at runtime and instantiate an object representing the service dynamically. This allows them to generate notices to the developer based on the versioning extensions inside the WSDL file. This approach will not work for static clients who are generated at compile time and interact with the service based on the contract generated once. To allow these clients to be able to retrieve any versioning information that is added to the WSDL description after they are generated new extensions should be developed that are added to the messages received when interacting with the web service itself. To achieve this, these extensions will have to be designed and incorporated to the tools used to publish a web service and encode the messages sent to the clients like Axis2 [9].

6. CONCLUSIONS
This paper has identified the requirements that should be put on solutions providing versioning information to the users of web services. The most critical requirement is full backwards compatibility: adding versioning information should at no time break existing consumers. At maximum information should be delivered about the current and future versions, but an description should not have to be updated more than once. Finally the provider should be able to give hints to the consumer on which operations to use and to avoid.

Section 3 discussed two other approaches to adding version information to web services. Both approaches have been discussed and reviewed with regards to the requirements stated in section 2. Most state of the art approaches are based on registries holding information on the relations and differences between versions and are responsible for communicating these to consumers. The WSDL descriptions therefore do not deliver any extra information to the consumer. An approach very similar to the one in this paper is WSDL versioning. The resulting WSDL descriptions of this method cannot be used by consumers that are unaware of the versioning extension and can only be used on adapted servers. This approach seems only viable in big enterprise environments.

This paper has shown how to extend WSDL files with in-line versioning information. With these extensions the client of a web service can be informed of deprecated methods, guaranteed methods and be referred to newer versions of the web service. Through the insertion of a minimal number of tags in an existing WSDL file this can be enriched with versioning information without breaking existing consumers when interacting with the service.

Future work
The extensions proposed in section four can be added to WSDL files manually after launching a new version of a web services. The extensions are designed to be as compact and easy to use as possible, but still there is some work to be done by the developer. Tool support should be developed to automatically generate the proposed extension from language specific notations like Java annotations. This way existing notations can be used to automatically extend the availability of versioning information to the consumer of a web service, without any extra effort by the provider.

Our approach is to make information that most likely already exists in the source code of objects deployed as web service available to service consumers. Our extensions are designed to deliver versioning information to the consumer, but there might be more information that can be transported to the service consumer through WSDL extensions like for example pre- or postconditions for operations. Further research into this area might reveal more information that can be added to WSDL descriptions, delivering more meta-data to the service consumer without burdening the service provider.

7. ACKNOWLEDGMENTS
I would like to thank Maarten Fokkinga and my fellowstudents in the “Data Application and Integration” track for their valuable input.

8. REFERENCES
