On Dependability of Composite Web Services with Components Upgraded Online

Vyacheslav Kharchenko
Department of Computer Systems and Networks
National Aerospace University, Kharkiv
Ukraine

Peter Popov
Centre for Software Reliability
City University, London
UK

Alexander Romanovsky
School of Computing Science
University of Newcastle
Newcastle upon Tyne
UK

Abstract

Ensuring dependability of composite Web services, dynamically composed of component Web services, is an open issue. One of the main difficulties here is due to the fact that component Web services can and will be upgraded online. The challenge is then to ensure that the overall dependability of the composite service is not undermined. The solutions we propose in this position paper make use of natural redundancy present in systems containing a new and an old release of the component.

1. Introduction

The Web service architecture [1] is rapidly becoming the de facto standard environment for achieving interoperability between different software applications running on a variety of platforms. This architecture supports development and deployment of open systems in which component discovery and system integration can be postponed until the systems are executed. The individual components (i.e. Web Services - WSs) advertise their services via a registry (typically developed using the UDDI standard[1]) in which their descriptions, given in a standard XML-based language called Web Service Definition Language (WSDL[2]), can be looked up. After a WS, capable of delivering the required service, has been found it can be used or even dynamically integrated into a composite WS.

The WS architecture is in effect a further step in the evolution of the well-known component-based system development with off-the-shelves (OTS) components. The main advances enabling this architecture have been made by the standardisation of the integration process (cf a set of interrelated standards such as SOAP, WSDL, UDDI, etc.). WSs are the OTS components for which a standard way of advertising their functionality has been widely adopted.

The WS architecture is now extensively used in developing various critical applications such as banking, auctions, internet shopping, hotel/car/flight/train reservation and booking, e-business, e-science, business account management. This is why ensuring dependability in this architecture is an emerging area of research and development [1], [2]. The need for measures towards providing the users of WS with information about WS dependability is discussed briefly in a recent report [3] in which the idea of 'Service Management' is outlined through a set of capabilities such as 'monitoring, controlling, and reporting of service qualities and service usage'.

The problem of dealing with online system upgrades is well known and a number of solutions have been proposed (see, for example [4]). The main reasons for upgrading systems are improving/adding functionality or correction of bugs. The difficulties in dealing with upgrades of COTS components in a dependable way are well recognised and a number of solutions have been proposed. The WS architecture poses a new set of problems manly caused by its openness and by the fact that the component WSs are executed in different management domains and are outside of the control of the composite WS. Moreover, switching such systems off or inflicting any serious interruptions in the service they provide is not acceptable, so all upgrades have to be dealt with seamlessly and online.

One of the motivations for our work is that ensuring and assessing dependability of complex WSs is complicated when any component can be replaced online by a new one with unknown dependability characteristics. There is clearly a need to develop solutions making use of the natural redundancy existing in such systems and guaranteeing that the overall dependability of the composite system is improving rather than deteriorating. Note that the idea of using the old and the new release of a program side by side to improve its dependability is clearly not new, it was first mentioned by B. Randell in his work on recovery blocks in which the earlier releases of the primary alternate are seen as a source of secondary alternates [5].

In this paper we discuss how a dependability measure – the confidence in the correctness of a WS – can be integrated in both – non-composite and composite WSs. In detail, section 2 gives an informal overview of the confidence in the correctness of a WS and how it can be published alongside the WS functionality. In section 3 we discuss the effect of WS upgrade on a composite service which depends on other upgraded WSs and in section 4 we discuss how the upgrade may affect the confidence in the correctness of the component service.

[1] http://www.uddi.org/

[2] http://www.w3.org/TR/wsdl

2. Confidence in WS correctness

WS, as any other complex software may contain faults which may manifest themselves in operation. In many cases the users of the WSs may benefit from knowing how confident they can be in the correctness of the information processing provided by the WSs. This issue may seem new in the context of WSs but is not new for some well-established domains with high dependability needs such as safety critical applications for which it is not unusual to state dependability requirements in probabilistic terms, e.g. as probability of software failure per demand. This fits nicely in the context of WSs, which can be seen as successive invocations of the operations published by a WS. It may be very difficult (or impossible) to guarantee that software behind a WS is flawless, but the confidence of the consumers will, no doubt, be affected by knowing for how long the service has been in operation and by how many failures have been observed. Informally, I will be much more confident in the results I get from a piece of software after I have seen it in operation for a long period of time without a failure than if I have not seen it in operation at all. How long software has been used is no guarantee that I will have high confidence in its correctness. Clearly, if I have seen it fail many times in the past I will take with doubts the next result that I get from this piece of software.

Building confidence in the correctness of WS can be formalised. Bayesian inference [6] is a mathematically sound way of expressing the confidence combining the knowledge about how good or poor the service is prior to deployment with the empirical evidence which becomes available after deployment. A priori knowledge can be calculated by the WS provider using standard techniques for reliability assessment, e.g. the quality of the development process or other techniques such as [7].

2.1. 'Publishing' confidence in WS correctness

Here we omit the details about how the confidence can be calculated and concentrate, instead, on practical ways of 'publishing' this confidence (or indeed any other dependability related measure) using the adopted standard for WSs. For simplicity, consider, that the confidence is a floating point number[3] which gets recalculated every time the service is called upon and which we would like to make available on demand. To illustrate the idea let us assume that the following is a fragment of the WSDL description of a WS:

```
<types>
  <s:schema … >
    <s:element name="Operation1Request">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
               name="param1" type="s:int">
          <s:element minOccurs="0" maxOccurs="1"
               name="param2" type="s:string">
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="Operation1Response">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
               name="Op1Result" type="s:string">
        </s:sequence>
      </s:complexType>
    </s:element>
    …
</types>
```

In other words, the WS interface publishes an operation `operation1` which requires two parameters when invoked, `param1` of type `int` and `param2` of type `string`, and returns a result `Op1Result` of type `string`.[4] Now assume that the WS provider wishes to 'publish' the calculated confidence in the correctness of `operation1`. There are two ways of doing it:

- the response to a consumer invoking `operation1` can be changed as follows:

```
<s:element name="Operation1Response">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
           name="Op1Result" type="s:string">
      <s:element minOccurs="0" maxOccurs="1"
           name="Op1Conf" type="s:double">
    </s:sequence>
  </s:complexType>
</s:element>
```

- a new operation is defined which takes as a parameter the name of an operation (for which the consumer seeks confidence) and returns the confidence in the quality of the operation:

```
<s:element name="OperationConfRequest">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
           name="operation" type="s:string">
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="OperationConfResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1"
           name="Op1Conf" type="s:double">
    </s:sequence>
  </s:complexType>
</s:element>
```

The advantage of the first implementation is that the confidence is associated with every execution of `operation1`. The obvious disadvantage is that the new WSDL description is not backward compatible with the old one, which is not acceptable for existing WS but may be OK for newly deployed services.

The advantage of the second solution is that the new

[3] Confidence is probability. When we say 'I am 90% confident that the result will be correct' the confidence is 0.9.

[4] For the sake of brevity the fragments of the WSDL description related to messages, parts and the service are not shown.

WSDL is backward compatible with the old WSDL. The disadvantage is that the confidence will have to be extracted in a separate invocation of a different operation published by the service (`OperationConf` in the example above), which may lead to complications.

Finally, a third option exists, which combines the advantages of both solutions given above. It consists of defining a new operation, e.g. `operation1Conf`, in which the response is extended by a number providing the confidence in the correctness of the operation. This approach allows the 'confidence conscious' consumers to switch to using `operation1Conf`, while it does not break the existing client applications which can continue to use `operation1`, i.e. backward compatibility is achieved.

2.2. Confidence in the correctness of a composite WS

Now, suppose that a composite service has been deployed, which depends on two other WSs, Web-Service 1 and Web-Service 2, as depicted in Fig. 1. The confidence in the quality of the composite service will be affected by the confidence in the quality of the two WSs it depends on and by the confidence in the quality of the composition (the design of the composition and its implementation, i.e. the "glue" code held in the composite WS itself). The approach presented above for publishing the confidence in the correctness of a WS is directly applicable to composite services, too. The difference between a non-composite and a composite service is merely in the way the confidence is calculated. For a non-composite service a standard textbook Bayesian inference can be used, while for a composite WS an approach presented in a recent paper, [8], can be used.

3. WS Online Upgrading

The fact that the composite WS depends on third-party services poses a problem, which is well-known for any component-based software development with OTS components. When a new release of an OTS component is made available the system integrator has two options:

- change their 'integrated' solution[5] to use the new release of the OTS component. This may cause problems for the integrated solution which may require significant effort to rectify.
- stick to the old version of the OTS component and take the risk to face the consequences if the vendor of the OTS component ceases to support the old releases of the OTS component.

The challenge here is to develop solutions supporting these options in an open environment when the new releases of WSs are becoming operational online. In this work we assume that the old and new releases of the component have the same functionality and there is backward compatibility between the two releases, i.e. the interface of the old release is a subset of the operations published by the new interface, which is typical for the WS architecture.

The situation with a composite WS is very similar to the one with any other OTS software component. Indeed the Web-Service 1 and Web-service 2 in Fig. 1 are two components integrated in the composite WS; conceptually this is equivalent to integrating any other OTS software component in an integrated solution. There may, however, be a difference from the point of view of maintenance between a composite WS and an integrated solution based on OTS components. In the latter case, as indicated above, the integrator has a choice whether to update the integrated solution with every new release of the OTS components or not. Such a choice may not exist in the former case of composite WSs. The deployment of a composite WS assumes that the WSs used by the composite service (Web-Service 1 and Web-Service 2 in our example in Fig. 1) have been deployed by their respective providers. If the providers decide to bring down their services the composite service may become unavailable, too. What seems more interesting is that when the provider of a service on which the composite service depends decides to update their service the provider of the composite service may not be even notified about the update. The composite service may be affected without its provider being able to do anything to prevent this from happening. Thus the provider of the composite WS is automatically locked-in by the very decision to depend on another WS.

Are there ways out of the lock-in? If not, can the provider of the composite WS do something at least to make the users of the composite WS aware of the potential problems as a result of the update(s) beyond their control? Below we discuss two plausible alternatives. These may be used by a provider of a composite service in case that the WSs used in the composite service do not publish confidence in their correctness explicitly as we described in section 2.

3.1. Several releases of the same WS are operational

This scenario is depicted in Fig. 2. The choice of whether to switch to a new release of a WS used by the composite service is with the provider of the composite WS. The provider of the composite service may use whatever methods are available to them to assess the 'quality' of the new release before deciding whether or not to move to the upgraded version(s) of the used WS.

The designer of the composite service may even make provisions at design stage of the composite WS which facilitates the assessment of the new releases of the services the composite service depends on when these become available. An example of such design would be making it possible to run 'back-to-back' the old and the new releases of the WS used in the composite service. During the transitional period (when the new release, WS 1.1 in Fig. 2, becomes available) the old version will continue to be the

[5] A term used by ECUA: http://www.esi.es/en/Projects/ecua/ecua.htm

version used by the composite WS, but by comparing the results coming from the old and the new release, WS 1.0 and WS 1.1 respectively, the provider of the composite WS will gain empirical evidence about how good the new release, WS 1.1, is. Once the composite service gains sufficient confidence in WS 1.1 it may switch to using it and cease using WS 1.0. Essentially, the composite service will have to run on its own a 'testing campaign' against the new release of the WS and may use the old release as an 'oracle' in judging if WS 1.1 returns correct results.

We have reported elsewhere in a different context how Bayesian inference can be used to assess the confidence in reliability of fault-tolerant software [9]. The same approach is directly applicable in the context of Fig 2 and the confidence in the composite service can be published, as described in section 2.

3.2. Only the latest release of a WS is operational

Under this scenario Fig. 1 remains applicable: the most recent release of Web Service 1 will be deployed behind the interface WS 1. The options left to the provider of the composite service are very limited. If the new release is at least distinguishable from the previous release, e.g. the release carries the release number (in the example above 1.0 will be replaced by 1.1) the provider of the composite service will be able to 'adjust' the confidence in the quality of the composite service, published to its consumers. A conservative view would be to reduce the confidence in the quality of the service every time a new release is made available compared with the confidence achieved with the old release of the WS 1 and recalculate the confidence in the composite service accordingly. A discussion of how the confidence can be 'recalculated' has been presented elsewhere [8].

4. Architectural Solutions on the Component WS

Disciplined and systematic dealing with WS upgrades starts with the way in which the WS providers internally deal with this issue. This is crucial for the confidence in the correctness of the composite service which uses these WSs. In this section we discuss solutions for smooth and dependable upgrades to be applied internally on the WS side. Clearly in this developments the WS should not be treated as a black box or as a COTS item. The WS manager should decide when to employ a new release and how to do this. Our solutions offer a systematic way of doing this online.

The underlying assumption here is that the WS is accessed through the interface published in a registry and the transition from the old release to the new one should be transparent for the consumer of the service. Simple replacement of an old version with a new one is a risky approach because it is difficult to be sure that the new release is no worse than the old one. We believe that all WS providers should always deploy their WSs in a special environment which has features for transparent upgrade including: interactive features for monitoring the dependability of old and new versions (including typical adjudicator functionality for comparing their results), a support for several modes of operations (one version, old and new versions in parallel, complete switch to a new version) and a standard interface corresponding to the WSDL description of the WS.

The WS provider should be able to monitor the way the new WS is operating and choose the best way of ensuring the dependability of his/her service. The main difference between this approach and the approaches outlined in sections 2-3 is that the old and the new versions are developed in house and the provider has much more information available about their dependability characteristics. Moreover it is possible to correct a new version and use various sophisticated and specific means of error detection. In addition to this it is possible to apply very adaptive and flexible ways of employing diversity: recovery blocks, warm and cold reservation, self-checked pair, etc.

We are now working on a first prototype implementation of such an environment using our extensive work on dependable WS composition within DSoS (Dependable Systems of Systems) IST project [10]. To ensure dependability of this implementation we use a number of local computers to implement a WS. These nodes are connected via RMI. Java server pages (JSPs) are used at the front-end of the service which allows for an easy integration using RMI with the rest of the Java infrastructure.

5. Discussion

In the context of solutions at the level of composite WSs discussed in sections 2-3 one problem requires careful consideration: the problem of dynamic notification of the WS clients about releasing of a new version. We have not discussed the ways in which the composite WSs are informed about this or the ways in which they get a reference to a new release of a component WS. There are several degrees of notification and various ways to implement it. One possibility is to use existing registry mechanism and extend the WSDL description of a WS to add a reference to a new release of a WS: this will allow a client to detect this with both releases staying operational. Another possibility is to use a WS notification service[6] as a separate mechanism to inform all the clients of a WS about a new release. A similar approach would be to explicitly notify the subscribers (clients) using some form of "call-back" function to the composite WS. Note that in the context of the "confidence" we have already assumed some provision for making the composite service aware of the change. Normally, the composite WS will see a "drop" in

[6]http://www-106.ibm.com/developerworks/webservices/library/specification/ws-notification/

the confidence as a result of the upgrade if the provider of the WS used by the composite service publishes only the version of the update. The updated confidence will be calculated more accurately if the services used by the composite service publish the confidence in the correctness of the new releases.

We are at present pursuing this research in several directions. First of all, we are developing a taxonomy of WS faults that will take into consideration specific characteristics of WSs and failures which can occur during their upgrade, as well as a set of metrics for assessing dependability of the old and new WS releases. Another strand of work is modelling architectural solutions and adjusting them online using a number of approaches to dependability assessment [11].

Acknowledgments

V. Kharchenko has been partially supported by the Royal Society grant (RS 16114). Popov and Romanovsky are partially supported by DOTS/EPSRC project.

References

1. Ferguson, D.F., T. Storey, et al., *Reliable, Transacted Web Services: Architecture and Composition*. 2003, Microsoft and IBM.
2. Tartanoglu, F., V. Issarny, et al., *Dependability in the Web Service Architecture*, in *Architecting Depndable Systems*. 2003, Springer-Verlag. p. 89-108.
3. Group, W.C.W., *Web Services Architecture*. 2004.
4. Romanovsky, A. and I. Smith. *Dependable On-line Upgrading of Distributed Systems*. in *COMPSAC'2002*. 2002. Oxford. p. 975-976.
5. Randell, B., *System Structure for Software Fault Tolerance.* IEEE Transactions on Software Engineering, 1975. **SE-1**(2): p. 220-232.
6. Box, G.E.P. and G.C. Tiao, *Bayesian Inference in Statistical Analysis*. 1973: Addison-Wesley Inc. 588.
7. Littlewood, B. and D. Wright, *Some conservative stopping rules for the operational testing of safety-critical software.* IEEE Transactions on Software Engineering, 1997. **23**(11): p. 673-683.
8. Popov, P. *Reliability Assessment of Legacy Safety-Critical Systems Upgraded with Off-the-Shelf Components*. in *SAFECOMP'2002*. 2002. Catania, Italy: Springer-Verlag. p. 139-150.
9. Littlewood, B., P. Popov and L. Strigini. *Assessment of the Reliability of Fault-Tolerant Software: a Bayesian Approach*. in *19th International Conference on Computer Safety, Reliability and Security, SAFECOMP'2000*. 2000. Rotterdam, the Netherlands: Springer.
10. Romanovsky, A., P. Periorellis and A.F. Zorzo. *Structuring Integrated Web Applications for Fault Tolerance*. in *6th International Symposium on Autonomous Decentralised Systems (ISADS 2003)*. 2003. Pisa, Italy. p. 99-106.
11. Kharchenko, V. *Methods of Estimation of Multiversion Safety Systems*. in *17th International System Safety Conference*. 1999. Orlando, USA. p. 347-352.

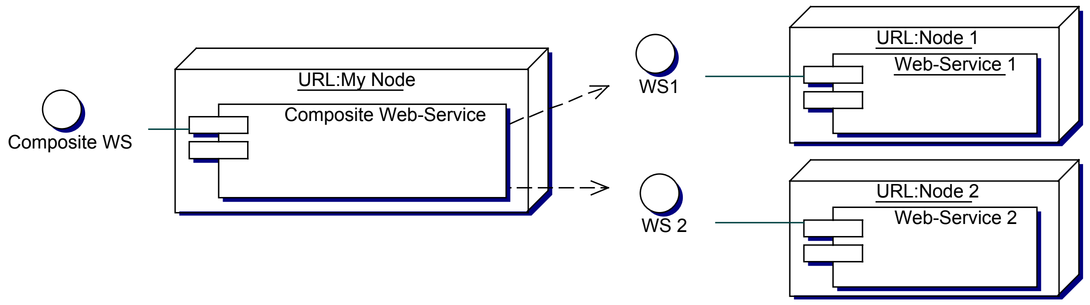

Fig. 1. A Deployment diagram of a composite WS which depends on two other WSs provided by third parties, Web-Service 1 and Web-Service 2, accordingly.

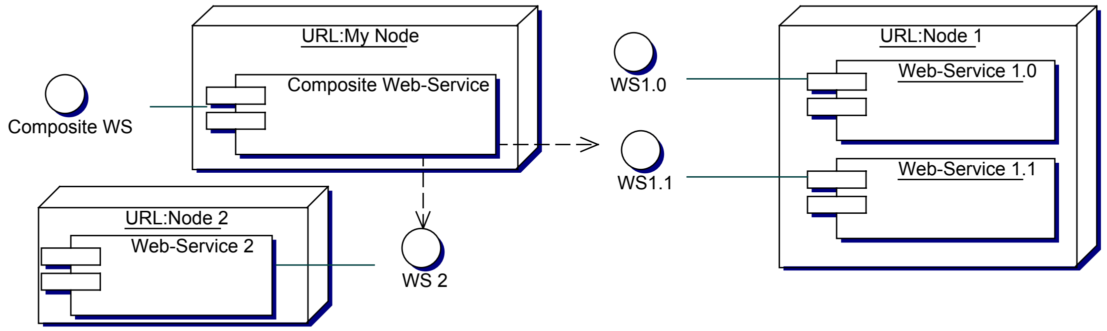

Fig. 2. A new release, Web-Service1.1, of a service is released, but the old version, Web-Service1.0, is also kept operational. The new release has no effect on the composite service, Composite Web-Service, which depends on the previous release as long as Web-Service1.0 is used. Eventually, the composite service is ‘upgraded’ to use the newer version, Web-Service1.1.