How to Enhance UDDI with Dependability Capabilities

Anatoliy Gorbenko[1] Alexander Romanovsky[2] Vyacheslav Kharchenko[1]

[1]*Department of Computer Systems and Networks*
National Aerospace University "KhAI"
17 Chkalov Str., Kharkiv, 61070, Ukraine
A.Gorbenko@csac.khai.edu
V.Kharchenko@khai.edu

[2]*School of Computing Science*
Newcastle University, Newcastle upon Tyne
NE1 7RU, UK
Alexander.Romanovsky@newcastle.ac.uk

Abstract

How dependability is to be assessed and ensured during Web Service operation and how unbiased and trusted mechanisms supporting this are to be developed are still open issues. This paper addresses the following questions: who should publish dependability parameters, in which way they should be distributed, and who (and how) should monitor these parameters in the global Service-Oriented Architecture. We discuss several techniques of on-line dependability monitoring and measurement, which extend the UDDI (Universal Description, Discovery and Integration) Business Registry with dependability metadata publishing and monitoring capabilities. The paper also proposes UDDI add-ons and light-weight user-side mechanisms for public operational and exceptional reporting.

1. Introduction

The concept of Service-Oriented Architecture (SOA) [1] was introduced in order to solve the problems of ensuring effective, reliable and secure interaction of open distributed systems consisting of autonomous and independently developed application components (services) deployed by different providers. Service interoperability in SOA is ensured through service interfaces being defined by common rules (the WSDL[1] description) and using a dedicated invocation mechanism (SOAP[2] messages). The descriptions of these components can be found by other software systems in a dedicated registry, and the components they invoked by means of XML-based messages transferred using Internet protocols. Achieving high dependability of SOA is crucial for a number of emerging and existing critical domains, such as telecommunication, Grid, e-science, e-business, etc.

SOA is unique in allowing access to a number of services with identical or similar functionalities, provided by different vendors and deployed to different platforms all over the Internet. In other words, SOA possesses the inherent redundancy and diversity of the existing Web Services [13]. To build dependable composite SOAs, users should be able to choose and use the most dependable components (i.e. Web Services) from the existing ones of similar functionality but diverse nature. Generally speaking, this can be achieved through UDDI[3] Business Registries (UBR), which provide information about particular Web Services and support service description, discovery and integration. However, as stated in [8], the existing service discovery mechanisms such as UDDI, UDDIe, WSIL, etc. are rudimentary in that the information stored is often not semantically rich enough for service differentiation. It would be a mistake to automatically consider a UDDI registry in its present state as an unbiased, trustworthy third party. Current public UBRs contain much of what is out of date, un-vetted and inconsistent. At the same time, Web Services clients should be able to make decisions about the Web services they will use at runtime, based on metadata attributed to those services [15]. Using UDDI metadata brings dynamic configuration to Web Services software architecture [11]. Web Service metadata might be associated with a set of additional schemas defined in WSDL or ontologies, like OWL (Web Ontology Language [16]). The W3C's Semantic Annotations for WSDL working group [12] has proposed a new annotation model whereby a ser-

[1] W3C, *Web Services Description Language (WSDL).* http://www.w3.org/2002/ws/desc

[2] W3C, *Simple Object Acess Protocol (SOAP).* http://www.w3.org/TR/soap12-part1

[3] W3C, *Universal Discovery, Description and Integration (UDDI).* http://www.uddi.org

0730-3157/08 $25.00 © 2008 IEEE
DOI 10.1109/COMPSAC.2008.33

vice provider can annotate interface elements with a small number of new WSDL extension attributes which represent references to concepts in an ontology.

There are a lot of studies describing various ontologies (*metadata*) related to a number of non-functional attributes of Web Services. Paper [4] provides an overview of resilience knowledge base (RKB) and Dependability and Security (D&S) ontology derived from the taxonomies of Avizienis et al. [6] and developed specifically for the RKB. This ontology is represented in OWL and incorporates 166 terms related to Dependability and Security, and 23 to Systems. A number of papers (e.g. [2, 3, 5, 9]) describe QoS attribute specification ontologies and QoS-aware discovery solutions based on service level agreements (SLAs). Papers [7, 8] also discuss attributes of performance, dependability (availability, reliability) and service cost as well as mechanisms of their aggregation.

Other dependability-related metadata that we propose including into the WSDL description is information about Web Service developers, implementation technology (i.e. *development metadata*); the hosting organization, location, deployment environment, network connection capacity, etc. (*deployment metadata*). Adding this meta-information will allow clients to decide how to use diverse WSs by decreasing common mode failures [14]. Finally, dynamic operational state parameters, such as current service load (the number of subscribers), CPU and memory usage, network loading, etc. might also be added to the extended WSDL description. Extending Web Service description with dependability metadata will bring us closer to a dependable Semantic Web [15]. However, the problem of providing information about service dependability that would be trustworthy from the client's point of view has yet to be solved.

In this paper we discuss solutions which enhance UDDI by implementing *dependable capability* (i.e. supporting *dependability metadata* dynamic publishing and monitoring as well as dependability-oriented service discovery). The main problem we are faced with is ensuring the trustworthiness and objectivity of dependability metadata. The rest of the paper is organised as follows. In section 2 we discuss the problem of enhancing UDDI by implementing dependability capability. Section 3 overviews Web Service dependability attributes and shows how they can be assessed. Sections 4 and 5 introduce our proposals concerning dependability attribute monitoring and dependability metadata publishing as well as analysing various monitoring techniques. Finally, section 6 briefly discusses the implementation of dependability capabilities.

2. UDDI Business Registry: key questions

UDDI Business Registries, which implement UDDI 2.0 or even current UDDI 3.0.2 specification [18], do not offer useful mechanisms for publishing and monitoring dependability metadata, and discovering dependability-based service, even though the idea of using such non-functional metadata for describing Web Services by extending their WSDL description is not new. A variety of Web Services metadata on QoS and dependability have been proposed in [2-5, 7-9], but these studies do not address *the problem of acquiring and estimating these attributes in a trustworthy and impartial manner*. Another key problem here is that WSDL and UDDI use static descriptions of services, whereas their dependability characteristics (e.g. availability, response time, etc.) can vary significantly during service operation.

Our vision is that the UDDI Business Registry has to act as kind of an impartial judge gathering trustworthy information about services dependability and disseminating it among services customers dynamically. Bellow we present a list of primary functions that have to be performed by an enhanced ('active') UDDI Business Registry. The term 'active' here means that UDDI Business Registry should perform activities aimed at evaluation and publishing of an actual dependability of Web Services registered (i.e. perform periodical monitoring, gather user's side feedbacks, update WS metadata, etc.):

1. Publishing dependability (and QoS) metadata.
2. Dependability-based service discovery.
3. Dynamic monitoring and gathering of dependability metadata.

Implementing the first two functions does not really pose serious difficulties whereas ensuring the third one still needs better understanding. The problem here is in ensuring the objectivity and trustworthiness of metadata acquisition. In this context we should answer the questions:

– Which dependability parameters are more important for users and which of them can be easily (and precisely) measured?

– Who should publish dependability parameters? – Service provider, customers or third parties (e.g. a UDDI Business Registry).

– Who (and how) should monitor these parameters in the global SOA? – Service provider, customers or third parties.

3. Which dependability attributes should be monitored and how it can be done?

Dependability of a computing system is the ability to deliver services, which can justifiably be trusted [6], at the proper time. According to this definition we have chosen the following dependability attributes, which are relevant to Web Services, and are easy to monitor during WS invocation: (i) availability; (ii) reliability; and (iii) response time (performance). There are several other attributes: describing QoS, service level agreements (SLAs) and dependability including authentication, confidentiality, non-repudiation, service cost, etc. [7], but we do not deal with them in this paper.

Service availability. The degree to which a service is operational and accessible when it is required for use determines its availability. System availability is a measure of the delivery of correct service with respect to the alternation of correct and incorrect services [6]. It also can be defined by a ratio of system's uptime to the overall execution time (including downtime). Unfortunately, such technique can hardly be applied for determining the availability of Web Services in a loosely coupled SOA. More adequately, the availability of a Web Service can be defined by the ratio of the total number of service invocations to the number of events when the service was unavailable (i.e. an exception "HTTP Status-Code (404): Not Found" was caught by client).

Service reliability. System reliability can be measured in terms of probability of failure-free operation, mean time between failures (MTBF) or failure rate. Reliability assessment of Web Services is a non-trivial problem. First of all, we should take into account service invocation rate which can vary widely. Another problem is that Web Service can return errors of two main types:

1. Evident erroneous response which results in an exception message. The probability of such errors can be measured by the proportion of the total number of service invocation to number of exception messages received (apart from exception "*HTTP Status-Code (404): Not Found*" that indicate service unavailability).
2. Non-evident erroneous response. It can be present in the form of incorrect data or calculation errors which do not entail immediate exception.

The last type of error is the most dangerous and can lead to unexpected program behaviour and unpredictable consequences. Detection of such errors is possible by comparing service response with response from another diverse service.

Service performance (response time). The service response time can be divided into (i) network delay time, (ii) connection waiting time and (iii) execution time. The execution time is the duration of performing service functionality, the connection waiting time is the time during request waiting in application server's queue, and, finally, network delay time is the delay of request transmissions between service consumer and provider. The network delay time can hardly be predicted due to uncertain network fluctuations whereas connection waiting time and execution time depend on service load and throughput. We propose a mechanism that can be used for approximate estimation of different parts of response time (see Fig. 1).

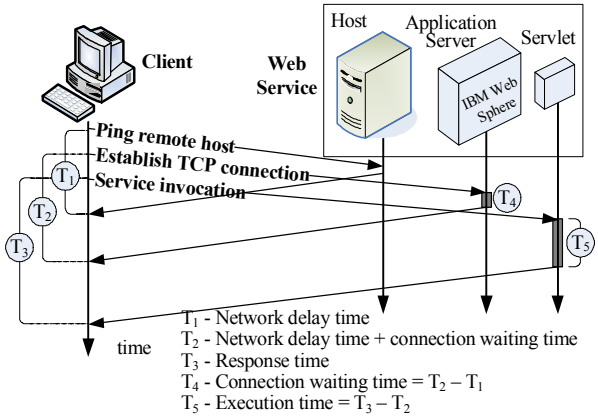

Figure 1. Service response time estimation

It includes three sequential operations which should be performed one after another without delay:

1. Pinging of a remote host (by sending the "ICMP Echo Request" message) to estimate network delay time.
2. Establishing a TCP connection with the application server to estimate network delay time together with connection waiting time.
3. Service invocation to estimate full response time.

4. Who should publish and monitor dependability parameters?

Dependability parameters, as well as the level of confidence in dependability [10] can be published by service owners in the form of a WSDL extension (see Fig. 2). However, this does not provide objectivity. The approach which will diminish bias is to entrust monitoring and publishing dependability parameters to a third party. It would be quite natural if we decided to use the UDDI Business Registry, which is in fact intended for services description, discovery and integration, as such third party.

```
<dependability>
  <availability>0.91</availability>
  <reliability>0.89</reliability>
  <response-time>
    <unit>millisecond</unit>
    <av>750</av>
    <min>135</min>
    <max>2100</max>
  </response-time>
...
</dependability>
```

Figure 2. WSDL extension describing dependability metadata

5. How to ensure the objectivity of dependability metadata?

There are several obvious restrictions under which the dependability monitoring has to be performed. First of all, the monitoring should not occupy plenty of service's operation time or resources. Secondly, the monitoring capability should be implemented without essential changing of existing client or service software. The other key question is how the objectivity and trustworthiness of the dependability metadata and monitoring results can be provided under existing limitations.

5.1. Direct dependability monitoring

The independence of dependability monitoring can be provided by granting this responsibility to the third parties. In [17] authors present a practical experience report on dependability monitoring of three diverse Bioinformatics Web Services performing similar BLAST[4] function. A mediator approach (set of intermediate monitoring services) was used to monitor WS dependability metadata and provide it for users. This work was a motivation for us to show i) that there are multiple similar WSs, ii) that they can be used simultaneously to achieve better dependability, iii) that this can be done by using metadata, iv) that this metadata can be collected at runtime.

It would be most natural if UDDI Business Registries together with providing services description (including dependability metadata) would also perform their direct monitoring with the purpose of guarantying trustworthiness of measured dependability attributes.

The problem here is, since such dependability attributes as "response time" or "availability" were measured at registry site, they would not be adequate for different clients dispersed all over the Internet. Besides, the accuracy of dependability estimation will be insufficient because it depends on inspection rate that can not be very high if we do not want to divert a service from handling true user's requests.

However, the direct monitoring can still be useful for providing the so-called "test of liveness": if a particular Web Service is unavailable for a long time, it should be removed from the UDDI as undeployed. This will help to maintain a UDDI Business Registry in an actual state.

5.2. User-side dependability monitoring and public reporting

There is no doubt that the most objective dependability estimation can be done only from the user's side. Here we present two light-weight mechanisms, providing user-side dependability monitoring coupled with reporting to the UDDI Business Registry (i.e. providing user's feedbacks). The idea of using such feedbacks is similar to the one described in [9].

1. Public exception reporting. User's application should straight away notify UDDI Business Registry about all exceptions catching during service invocation (Fig. 3).

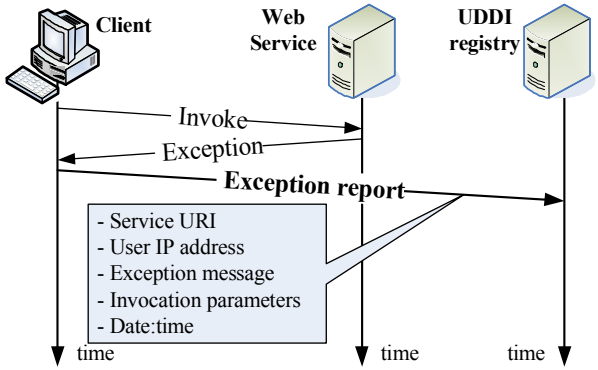

Figure 3. Public exception reporting

Exception report has to contain the URI (Uniform Resource Identifier) of a Web Service, user's IP address, exception message, service invocation parameters and invocation time.

Exception messages and stack traces can be used for determining the exact exception source (application software, Web Service middleware or network) [20]. This information can be used dynamically during the selection of the most suitable recovery technique.

2. Public operational reporting. After certain period of time (daily or weekly) or certain number of service invocations, user application sends an operation report to the UDDI Business Registry (Fig. 4) which contains the following information: Web Service URI (Uniform Resource Identifier) and user's IP; total number of service invocations; amount of cases when

[4] http://www.ncbi.nlm.nih.gov/blast/html/BLASThomehelp.html

the service was unavailable (the exception message was *"HTTP Status-Code (404): Not Found"*); total number of exceptions caught; the minimal, the maximal and average response times, etc. The results of dependability and performance monitoring of real e-Science WS as well as the monitoring technique used can be found in [17]. The UDDI Business Registry can utilize user's operational and exceptions reports to assess the Web Service usage and its dependability attributes, and to publish dependability metadata.

Taking into account locations (on the Internet) of a service and a particular client (based on service URI and user's IP address analysis) the UDDI Business Registry will be able to deal with network fluctuations and to assess a number of dependability attributes, including availability and average response time, for certain groups of users according to their locations on the Internet.

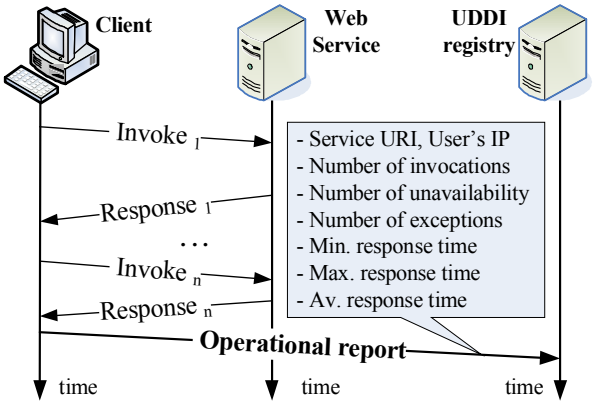

Figure 4. Public operational reporting

A deeper correlation analysis is also possible by means of applying data-mining technology. For example, all information gathering by UDDI is, undoubtedly, time sensitive and dependability assessment and prediction can/should be dependent on the time of the day/day of the week/etc. The information gathering by UDDI can also help clients to set the time-outs correctly.

6. Implementation

Additional functionality can be implemented in the UDDI as a set of plugins (add-ons), performing services monitoring, gathering user's feedback from operational and exceptions reports, assessing dependability attributes and performing further data mining. A common WSDL description has to be extended with new elements describing dependability metadata and other non-function characteristics. XML Schema (XSD) describing these elements and new namespace should also be specified.

An example of extending UDDI with new capabilities is described in [8]. The authors added new WSDL elements and implemented advanced service discovery functionality using JAXR[5], which is registry independent, so, theoretically, they find it easy to implement support for other discovery mechanisms. Our practical work centres in developing add-ons enhancing the open JUDDI repository [19]. These add-ons are pieces of Java program codes supporting the following:

1. Service dependability monitoring.
2. Gathering and processing user reports and exceptions reports.
3. Service dependability estimation and ranking, and further data-mining.
4. Publishing and updating dependability metadata.
5. Providing service discovery based on dependability metadata.

The monitoring functions should be implemented directly in the UDDI Business Registry or could be performed by specialized external service, which would periodically update dependability metadata in the UDDI. The code could also form the basis for an API to allow two-way interaction with users and dependability-based service discovery and selection, performed dynamically at runtime. In this work our goal is to avoid making any changes in the user's application. Therefore, the program code performing dependability monitoring and measurement on the client side and notifying the UDDI Business Registry *should be integrated directly* into the existing API for Web Services (AXIS[6], WSTK[7], SAAJ[8], etc.). Thus, the already existing application software should only be recompiled with new WSDK libraries. Of course, the programmers will have the ability to disable this new functionality if they are not interested in it.

7. Conclusions and Future Research

The existing UDDI specifications still have poor programming APIs which do not satisfy users' needs. In the paper we discuss a number of solutions called to enhance the UDDI with functionality for implementing dependability monitoring and publishing. Even though the proposed solutions are not widely accepted, it is

[5] Sun Microsystems, "JAXR", http://java.sun.com/xml/jaxr/index.jsp
[6] Apache eXtensible Interaction System (AXIS): http://ws.apache.org/axis/
[7] IBM Web Services Toolkit (WSTK): http://www.alphaworks.ibm.com/tech/webservicestoolkit/
[8] SOAP with Attachments API for Java (SAAJ): https://saaj.dev.java.net/

clear that they can be implemented as part of corporate Service Oriented Systems using private UDDI Business Registries. They also can be widely used in various Grid systems which rely on open Web Services Standards. Other useful capabilities which can be implemented in the UDDI are connected with more rigorous classification of the registered Web Services, supporting a search of alternative services with identical or similar functionality and providing trade-off between service cost and dependability. The service provider should be able to register several Web Services performing the same operations; such services are typically located in different Internet domains. The UDDI Business Registry has to return, to the user, a reference to the service which is the most reliable and the fastest for him to use. The next key question discussed in the paper is how often Web Services should be monitored by UDDI Business Registry. Our suggestion here is to use two-level UDDI. All new Web Services have to pass through the trial period during which they will be monitored quite often.

Acknowledgments. This work is partially supported by the EPSRC TrAmS platform grant and ICT FP7 DEPLOY IP.

10. References

[1] W3C, Web Services Architecture (2004): http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[2] Wang, Y., Vassileva, J. "Toward Trust and Reputation Based Web Service Selection: A Survey", *Proc. International Transactions on Systems Science and Applications (ITSSA) Journal*, vol 3, no. 2 (2007).

[3] Menasce, D.A.. "QoS Issue in Web Services". *IEEE Internet Computing*, vol. 6, issue 6 (2002), pp. 49-68

[4] T. Anderson, Z.H. Andrews, J.S. Fitzgerald, B. Randell, H. Glaser, I.C. Millard. "The ReSIST Resilience Knowledge Base". *Proc. The 37th Annual IEEE/IFIP International Conference on. Dependable Systems and Networks (DSN'2007)*, Supplemental Volume, Fast Abstract, Edinburgh, UK (2007)

[5] O'Sullivan, J., Edmond, D., Hofstede, A. "What is in a service? Towards Accurate Description of Non-Functional Service Properties", *Kluwer Academic Distributed and Parallel Databases*, vol 12 (2002), pp. 117-133

[6] Avizienis, J.-C. Laprie, B. Randell, C. Landwehr. "Basic Concepts and Taxonomy of Dependable and Secure Computing". *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1 (2004), pp. 11-33

[7] Yang, S., Lan, B., Chung, J.-Y. "Analysis of QoS Aware Web Services", *Proc. International Computer Symposium on Web Technologies and Information Security Workshop (ICS)* (2006)

[8] Lock, R., Dobson, G. "Developing an ontology for QoS", Dependability interdisciplinary research Collaboration (Internal Annual Project Conference), Nesc (National e-Science centre), Edinburgh (2005), pp. 128-132.

[9] R. Jurca, B. Faltings, W. Binder "Reliable QoS monitoring based on client feedback". *Proc. 16th international conference on World Wide Web* (2007), pp. 1003-1012.

[10] A. Gorbenko, V. Kharchenko, P.Popov, A. Romanovsky "Dependable Composite Web Services with Components Upgraded Online" / In R. de Lemos et al. (Eds.): Architecting Dependable Systems III, LNCS 3549. Berlin, Heidelberg: Springer-Verlag (2005), pp. 92-121.

[11] K. Januszewski "The Importance of Metadata: Reification, Categorization, and UDDI". Microsoft Corporation. http://msdn2.microsoft.com/en-us/library/ms953942.aspx

[12] W3C: *Semantic Annotations for WSDL (SAWSDL)*: www.w3.org/2002/ws/sawsdl

[13] A. Gorbenko, V. Kharchenko, A. Romanovsky. "Vertical and Horizontal Composition in Service-Oriented Architecture". Proc. *Workshop on Methods, Models and Tools for Fault Tolerance at IFM 2007*, Oxford, UK (2007).

[14] Lyu M.R. (edit.) Handbook of Software Reliability Engineering, McGraw-Hill Company, 1996, 805 p.

[15] W3C: *Semantic Web Activity: Advanced Development.* http://www.w3.org/2000/01/sw/

[16] W3C: *Web Ontology Language (OWL).* www.w3.org/2004/OWL/

[17] Y. Chen, A. Romanovsky "Improving the Dependability of Web Services Integration" *IT Professional: Technology Solutions for the Enterprise,* IEEE Computer Society, *issue* January/February (2008), pp. 20-26

[18] UDDI Version 3.0.2 Specification (2005): http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm

[19] Open source Java implementation of the UDDI specification for Web Services (JUDDI): http://ws.apache.org/juddi/

[20] A. Gorbenko, I.E. Komari, V. Kharchenko, A. Mikhaylichenko "Exception Analysis in Service-Oriented Architecture" / In H. C. Mayer, D. Karagiannis (eds.): Information Systems Technology and its Application, GI-Edition Lectures Notes in Informatics (LNI), P 107. GmbH, Bonn: Kölen Druck+Verlag (2007), pp. 228–233.