Experimenting with Exception Propagation Mechanisms in Service-Oriented Architecture

Anatoliy Gorbenko[1]
Alexander Romanovsky[2]
[1]Department of Computer Systems and Networks, National Aerospace University, Kharkiv, Ukraine
A.Gorbenko@csac.khai.edu
Alexander.Romanovsky@newcastle.ac.uk

Vyacheslav Kharchenko[1]
Alexey Mikhaylichenko[1]
[2]School of Computing Science, Newcastle University, Newcastle upon Tyne, UK
V.Kharchenko@khai.edu,
A.Mikhaylichenko@csac.khai.edu

ABSTRACT
Exception handling is one of the popular means used for improving dependability and supporting recovery in the Service-Oriented Architecture (SOA). This practical experience paper presents the results of error and fault injection into Web Services. We summarize our experiments with the SOA-specific exception handling features provided by the two development kits: the Sun Microsystems JAX-RPC and the IBM WebSphere Software Developer Kit for Web Services. The main focus of the paper is on analyzing exception propagation and performance as the major factors affecting fault tolerance (in, particular, error handling, and fault diagnosis) in Web Services.

Categories and Subject Descriptors
C2.4 [**Computer-Communication Networks**]: Distributed Systems – *Distributed applications*
D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms
Measurement, Performance, Design, Reliability, Experimentation, Languages

Keywords
Service-Oriented Architecture, dependability benchmarking, robustness, exception handling, exception propagation mechanisms, fault tolerance.

1. INTRODUCTION

The concept of service-oriented architecture (SOA) was introduced in order to solve the problems of ensuring effective, reliable and secure interaction of complex distributed systems. SOA assumes that such systems are constructed from loosely-coupled application modules (services) that have interfaces defined by common rules (the WSDL[1] description) and a dedicated invocation mechanism (SOAP[2] messages). The descriptions of these modules can be found by other software systems in a dedicated registry and the modules can then be invoked by means of XML-based messages transferred using Internet protocols.

Achieving high dependability of service-oriented architecture is crucial for a number of emerging and existing critical domains, such as telecommunication, grid, e-science, e-business, etc. Knowing the exact causes and sources of exceptions raising during operation of Web Service allows developers to apply the more suitable fault-tolerant [1] and error recovery techniques [2]. For example, paper [3] discusses two fault tolerance means, applicable in SOA: backward (based on rolling system components back to a previous correct state) and forward error recovery (which involves transforming system components into any correct state). The latter is usually application-specific and employs exception handling mechanisms. As backward error recovery is not always applicable for Web Services due to the simple fact that they cannot always be rolled back, exception handling is becoming a popular technique for ensuring fault-tolerance and error recovery of Web Services. In the practical experience report we present an experimental analysis of the SOA-specific exception propagation mechanisms and provide some insights into differences in error handling and propagation delays between two implementations of web services in IBM WebSphere SDK[3] and Sun Java application server SDK[4]. To provide such an analysis we have used fault injection technique. Fault injection is a well-proven method for assessing the dependability and fault-tolerance of a computing system. Although much work has been done in the area of fault injection and distributed systems in general, for example [4, 5], there appears to have been little research carried out on applying this to the SOA. Papers [6, 7] present a practical approach for the dependability benchmarking and evaluation of the robustness of Web Services. In particular, the authors of paper [7] describe a set

WEH'08, November 14, 2008, Atlanta, GA, USA.

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

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

[3] http://www-128.ibm.com/developerworks/webservices/wsdk/

[4] http://www.sun.com/software/products/appsrvr_pe/index.xml

of robustness tests (i.e., failures injection into web-services call parameters) which were applied during web-services execution in order to reveal possible robustness problems.

Papers [8, 9] introduce specialised ontologies used for systematic generation of fault, attack and latency injection test cases, and failure detection techniques for testing network packet loss and message corruption in the SOA. To summarise, we have found that the existing works above neither consider the propagation behaviour of the exceptions raised because of the injected faults nor study the performance with respect to the exception propagation caused by the use of different Web Services platforms.

The *objective* of the paper is to analyze the exception propagation mechanisms of the two Web Services development toolkits and understand their implications for performance of the SOA applications using them. Our experimental investigation was organised as shown on Fig.1.

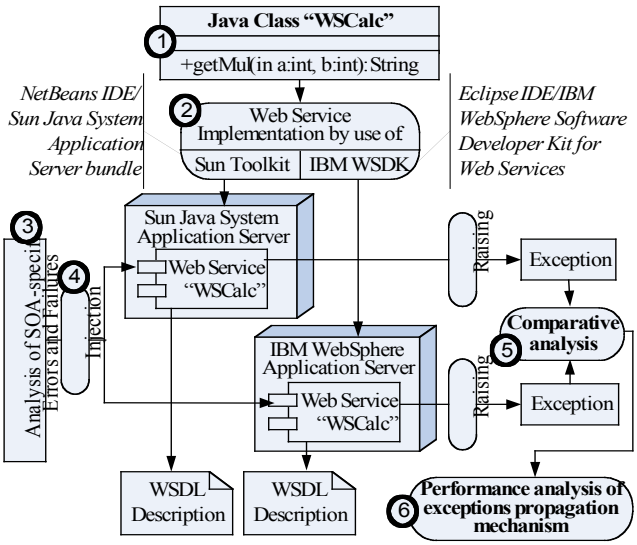

Figure 1. Research engineering process.

To conduct our experiments we first implemented as a Java class, WSCalc, which performs a simple arithmetic operation upon two integers, converting the result into a string (1). Then (2) we implemented two diverse Web Services using two different development toolkits described in section 2.

A brief description of the testbed WS is presented in section 3. In section 4, we analyse SOA-specific errors and failures (4) and inject them using our Web Service architecture (5). Section 5 reports the results of analysing and comparing the exception propagation mechanisms and performance implications (6).

2. DEVELOPMENT TOOLKITS

In our work we experimented with two widely used technologies: the Java cross-platform technology, developed by Sun and the IBM Web Service development environments and runtime application servers.

The reasons for this choice are that Sun develops most of the standards and reference implementations of Java Enterprise software whereas IBM is the largest enterprise software company.

2.1 NetBeans IDE/SJS Application Server

NetBeans IDE 5.0[5] is a powerful integrated environment for developing applications on the Java platform, supporting Web Services technologies through the Java Platform Enterprise Edition (J2EE). Sun Java System (SJS) Application Server is the Java EE implementation at Sun Microsystems.

NetBeans IDE with SJS Application Server support JSR-109, which is a development paradigm that is suited for J2EE development, based on JAX-RPC (JSR-101). Web Service functionality in NetBeans IDE is part of an end-to-end set of J2EE features. Also, NetBeans IDE provides wizards to create Web Services and Web Service's clients.

2.2 IBM WSDK for Web Service

IBM WebSphere Software Developer Kit Version 5.1 (WSDK) is an integrated kit for creating, discovering, invoking, and testing Web Services. WSDK V5.1 is based on WebSphere Application Server V5.0.2 and provides support for the following open industry standards: SOAP 1.1, WSDL 1.1, UDDI 2.0, JAX-RPC 1.0, EJB 2.0, Enterprise Web services 1.0, WSDL4J, UDDI4J, and WS-Security.

WSDK can be used with the Eclipse IDE. Eclipse provides a graphical interactive development environment for building and testing Java applications. WSDK adds to the standard Eclipse package the tools relating to Web Services, making it suitable for building Web Services. The required level of Eclipse is V2.1.1. The Eclipse package can be freely downloaded from the Eclipse Web site[6]. Supporting the latest specifications for Web Services WSDK enables to build, test, and deploy Web Services on industry-leading IBM WebSphere Application Server. Functionality of the WSDK V5.1 has been incorporated into the IBM WebSphere Studio family of products.

3. WEB SERVICE TESTBED

The starting point for developing a JAX-RPC Web Service is the coding of a service endpoint interface and an implementation class with public methods that must throw the java.rmi.RemoteException. To analyze features of the exception propagation mechanisms in the service-oriented architecture we have developed a testbed Web Service executing simple arithmetic operations. The implementation bean class of the Web Service providing arithmetic operations is shown in Fig. 2.

```
package ai.xai12.loony.wscalc;
public class WSCalc implements WSCalcSEI {
 public String getMul (int a, int b) {
 return new Integer(a * b).toString();
 }  ...   }
```

Figure 2. The implementation bean class of the Web Service providing arithmetic operations.

NetBeans IDE/SJS AppServer and Eclipse IDE/IBM WSDK support wizards that automatically generate service endpoint interface (SEI) and service description (WSDL-file) and deploy Web Service. However, in spite of the fact that both toolkits are based on the open specifications and interfaces we discovered a

[5] http://www.netbeans.org
[6] http://www.eclipse.org

sufficient number of differences in generated Web Service descriptions. They both require description of input and output parameters, definition of used namespaces (the default and target namespaces). At the same time, some prefixes and namespaces are defined but not used. There are some differences in the description of input and output parameters. As it will be shown below, these differences have some effect on exception propagation.

The testbed service was implemented by using two different development kits provided by Sun and IBM. Two diverse services obtained in such a way were deployed on the two hosts using different application servers: i) IBM WebSphere and ii) SJS AppServer. These hosts operated under Windows XP Profession Edition were located in the university's LAN. Thus, transfer delays and other network problems were insignificant and affected both testbed services in the same way.

4. INJECTION TECHNIQUE. SOA-SPECIFIC ERRORS AND FAILURES

In terms of the fundamental concepts of dependability, threats to computer systems include errors, faults and failures [10]. An error is that part of the system state that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service. A fault is the hypothesized cause of an error. Faults are usually classified into three major fault classes [10]: design faults, physical faults and interaction faults.

The main stages of the Web Services interactions are [11]: (i) service binding, (ii) service invocation, (iii) SOAP messaging, and (iv) requests processing by WS (Fig. 3). In our work we have experimented with 18 types of the SOA-specific errors and failures occurring during these stages (see Table 1) and dividing into three main categories: (i) network and remote system failures, (ii) internal WS errors and failures and (iii) client-side binding errors. They are general (not application specific) and can appear in any Web Service application during operation.

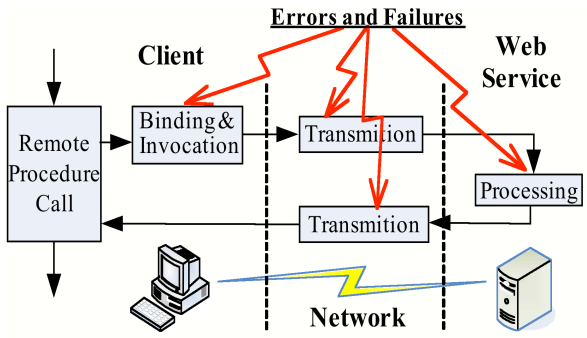

Figure 3. Errors and failures affecting on SOA.

We omitted in our measurement the stages of service discovering and integration (e.g. using the UDDI[7]) because their effect can be ignored as they are carried out only once before the sequences of other interactions.

[7] W3C, *Universal Discovery, Description and Integration*. http://www.uddi.org/

Network failures are unavoidable in the service-oriented architecture due to global distribution of its components. We analysed network connection break-off at the client-side and remote host unavailability when it is off-line or unreachable due to network failures.

Exceptions are manifestations of a symptom, i.e. an error or a failure occurred [15]. In our work we investigated a reaction of two WS platforms (middleware) provided by Sun Microsystems and IBM on the injected errors and faults in order to answer the questions like "Is the WS middleware robust to errors in invocation parameters?", "Do the exception stack traces provide exact information about the root causes of exceptions?", "Whether the exceptions propagation chain and propagation velocity (performance) depend on WS middleware used?".

Table 1. SOA-specific errors and failures

№	Type of error/failure	Error/failure domain
1.	Network connection break-off	Network and system failures
2.	Domain Name System is down	
3.	Loss of request/response packet	
4.	Remote host unavailable	
5.	Application Server is down	
6.	Suspension of WS during transaction	Service errors and failures
7.	System run-time error	
8.	Application run-time error	
9.	Error causing user-defined exception	
10.	Error in Target Name Space	Client-side binding errors
11.	Error in Web Service name	
12.	Error in service port name	
13.	Error in service operation's name	
14.	Output parameter type mismatch	
15.	Input parameter type mismatch	
16.	Error in name of input parameter	
17.	Mismatching of number of input params	
18.	WS style mismatching ("Rpc" or "Doc")	

Common-case network failures are down state of DNS or packets lost due to the network congestion. Besides, the operation of Web Service depends on the operation of the system software like web-server, application server and database management system. In our work we analysed failures occurring when the application servers (WebSphere or SJS AppServer) were shut down. Client errors in early binding or dynamic interface invocation (DII) (like "Error in Target Name Space", "Error in Web Service name", etc.) occur because of the changes in the invocation parameters, and/or inconsistencies between the WSDL-description and the service interface. Finally, the service failures are connected with program faults and run-time errors causing system- or user-defined exceptions. System run-time errors like "Stack overflow" or "Lack of memory" result in the exceptions at the system level as a whole. Operation "Division by zero" is also caught and generates an exception at the system level but it is easier to simulate such system error than other ones.

The typical examples of the application run-time errors are "Operand type mismatch", "Product overflow" and "Index out of bounds". In our experiments we injected the "Operand type mismatch" error, hangs of the WS due to its program getting into a loop and error causing user-defined exception (exception defined by a programmer during WS development).

Service failures (6, 7, 8) were simulated by fault injection at the service side. Client-side binding errors (10-18) which are, in fact, a set of robustness tests (i.e., invalid web-services call parameters) were applied during web-services invocation in order to reveal possible robustness problems in the web-services middleware. We used a compile-time injection technique [13] where a source code is modified to inject simulated errors and faults into the system. Network and system failures were simulated by shutting down manually of DNS server, application server and network connections at the client and service sides.

5. ANALYSIS OF EXCEPTION PROPAGATION MECHANISMS AND PERFORMANCE IMPLICATIONS

To analyze features of exception propagation mechanisms and performance implications in SOA depending on the Web Services development toolkit used, we inject errors in the testbed services and client applications, and also simulated network failures.

5.1 Exceptions Correspondence Analysis

The experiments were carried away with simple Web Services executing arithmetic operations which were deployed on two application services: SJS AppServer and IBM WebSphere. Some results of our experiments with the Web Services exceptions are shown in Table 2 which describes a relationship between errors/failures and the exceptions raised at the top level on different application platforms. The full stack traces and technical details can be found in [12]. As it was discovered, some injected errors and failures cause the same exception so we were not always able to define the precise exception cause. There are several groups of such errors and failures: 1 and 2 (Sun); 3 and 6 (Sun); 4 and 5 (Sun); 1, 2 and 5 (IBM); 3 and 6 (IBM).

Some client-side binding errors (11 – "Error in Web Service name", 12 – "Error in service port name") neither raise exceptions nor affect the service output. This happens because the WS is actually invoked by the address location, whereas the service and port names are only used as supplementary information. Moreover, the WS developed by using IBM WSDK and deployed on the IBM WebSphere application server, tolerates such binding errors internally: 10 - "Error in Target Name Space", 14 - "Output parameter type mismatch", and 16 - "Error in name of input parameter". These features are supported by the WSDL description and a built-in function of automatic type conversion.

Errors in the name of the input parameter were tolerated because checking the order of parameters has a priority over the coincidence of parameter names in the IBM implementation of Web Service. On the other hand it seems like Websphere is unable to detect a potentially dangerous situation resulted from the parameters mishmash.

5.2 Exception Propagation and Performance Analysis

Table 2 shows the exceptions raised at the top level on client's side. However, a particular exception can be wrapped dozens of times before it finally propagates to the top. This process takes time and significantly reduces performance of exception handling in service-oriented architecture. Fig. 4 shows a fragment of Java code that follows a critical section of program, catches any exceptions and prints an exception stack trace in case of error occurrence.

```
...
catch (Exception e) {
  e.printStackTrace();
}
```

Figure 4. Example of client Java-code that prints the stack trace.

An example of the stack trace corresponding to "Operand Type Mismatch" run-time error caught by Web-Service is given in the Fig. 5. The exception propagation chain has four nested calls (started with "at" preposition) in case of using WS development kit from Sun Micro-systems. For comparison, the stack trace of IBM-based implementation has 63 nested calls for the same error.

```
java.rmi.ServerException: JAXRPC.TIE.04:
Internal Server Error (JAXRPCTIE01:
java.lang. NumberFormatException: For input
string: "578ER")

at com.sun.xml.rpc.client.dii.BasicCall.
invoke(BasicCall.java:497)

at ai.c1.xai12.wstest.InvoceWS.invoce
(InvoceWS.java:125)

at ai.c1.xai12.wstest.InvoceWS.
invoceByVector(InvoceWS.java:75)

at wstest.Main.main(Main.java:42)
```

Figure 5. Stack trace of failure No 8, raised in the client application developed in NetBeans IDE by using JAX-RPC implementation of Sun Microsystems.

The results of exception propagation and performance analysis are represented in Table 3. This table includes a number of exceptions stack trace (length of exceptions propagation chain, i.e. the count of different stack traces for this particular failure) and propagation delay (min, max and average values) which is a time between the invocation of a service and capture of the exception by a 'catch' block. As can be seen from Table 3, the IBM implementation of the Web Service has almost twice as good a performance as that of the service implemented in the Sun technology.

The performance of exception propagation mechanisms has been monitored at the university LAN on heterogeneous server platforms. The first row of the table corresponds to the correct service output without any exceptions. The rows, marked in bold, correspond to the cases of correct service outputs without exceptions in spite of injected errors. It is evident from the table, that exceptions propagation delay is several times greater than working time. However, the exception propagation delay of the Web Service, developed with NetBeans IDE using JAX-RPC implementation of Sun Microsystems, was two times shorter than the delay we had when we used IBM WSDK. It explains the fact that the exception propagation chain in the IBM implementation of the Web Service is, usually, much longer.

The factors affecting the performance and differences between the two web-service development environments most likely depend on the internal structure of toolkits and the application servers used. We believe that the most likely reason for this behaviour is that JAX-RPC implementation by Sun Microsystems has larger number of nested call in contrast to IBM WSDK.

Table 2. Example of top-level exceptions raised by different types of errors and failures

Type of error/failure	Exception message at using Sun Microsystems WS Toolkit	Exception message at using IBM WS Toolkit (WSDK)
Network connection break-off; DNS is down	"HTTP transport error: java.net. UnknownHostException: c1.xai12.ai"	"{http://websphere.ibm.com/webservices/} Server.generalException"
Remote host unavailable (off-line)	"HTTP Status-Code 404: Not Found - /WS/ WSCalc"	"{http://websphere.ibm. com/webservices/} HTTP faultString: (404)Not Found"
Suspension of Web Service during transaction	*Waiting for response during too much time (more than 2 hours) without exception*	"{http://websphere.ibm.com/webservices/} Server.generalException faultString: java.io. Interrupted IOException:Read timed out"
System run-time error ("Division by Zero")	"java.rmi.ServerException: JAXRPC.TIE.04: nternal Server Error (JAXRPCTIE01: caught exception while handling request: java.lang. ArithmeticException: / by zero)"	"{http://websphere.ibm.com/webservices/} Server.generalException faultString: java.lang.ArithmeticException: / by zero"
Application error causing user-defined exception	"java.rmi.RemoteException: ai.c1.loony.exception. UserException"	"{http://websphere.ibm.com/webservices/}Server. generalException faultString:(13)UserException"
Error in name of input parameter	"java.rmi.RemoteException: JAXRPCTIE01: unexpe-cted element name:expected=Integer_2, actual=Integer_1"	*OK - Correct output without exception*

Table 3. Performance analysis of exceptions propagation mechanism

	WS Development Toolkit	NetBeans IDE (Sun)				IBM WSDK			
№	Type of error/failure	no of stack traces	exception's propagation delay, ms			no of stack traces	exception's propagation delay, ms		
			min	max	av.		min	max	av.
	Without Error/Failure	*0*	*40*	*210*	*95*	*0*	*15*	*120*	*45*
1.	Network connection break-off	38	10	30	23	16	10	40	28
2.	Domain Name System is down	28	16	32	27	16	15	47	34
3.	Loss of packet with client request or service response	-	>7200000			15	300503	300661	300622
4.	Remote host unavailable (off-line)	9	110	750	387	11	120	580	350
5.	Application Server is down	9	70	456	259	16	100	550	287
6.	Suspension of Web Service during transaction (getting into a loop)	-	>7200000			15	300533	300771	300642
7.	System run-time error ("Division by Zero")	7	90	621	250	62	120	551	401
8.	Calculation run-time error ("Operand Type Mismatch")	4	90	170	145	63	130	581	324
9.	Application error causing user-defined exception	4	100	215	175	61	150	701	366
10.	Error in Target Name Space	4	100	281	180	***0***	***10***	***105***	***38***
11.	*Error in Web Service name*	***0***	***40***	***120***	***80***	***0***	***10***	***125***	***41***
12.	*Error in service port name*	***0***	***30***	***185***	***85***	***0***	***15***	***137***	***53***
13.	Error in service operation name	4	90	270	150	58	190	511	380
14.	Output parameter type mismatch	14	80	198	160	***0***	***15***	***134***	***48***
15.	Input parameter type mismatch	4	80	190	150	76	90	761	305
16.	Error in name of input parameter	4	70	201	141	***0***	***10***	***150***	***47***
17.	Mismatching of number of input service parameters	4	80	270	160	61	130	681	350
18.	Web Service style mismatching	4	70	350	187	58	90	541	298

The fact that that the service client, developed using the Sun WS toolkit, does not raise any exception even after more than 2 hours of waiting in cases of service suspension or packets loss results in retarded recovery action and complicates the developers' job. Analysis of the exception stack trace and propagation delay can help in identifying the source of the exception.

For example, failures 1 - "Network connection break-off" and 2 - "Domain Name System (DNS) is down" raise the same top-level exception "*HTTP transport error: java.net.UnknownHostException: loony.xai12.ai*". However, if we use the Sun WS toolkit we can distinguish these failures by comparing numbers of the stack traces (38 vs. 28). If we use IBM WSDK we are able to distinguish between failure 5 – "Application Server is down" and failures 1 and 2 on the basis of analysis of the exception propagation delay (it is one order greater).

Fig. 6 below shows the classification of errors and failures taking into consideration their sources, consequences, correspondence analysis of exceptions raised by them (see section 5) and also possible means for recovering and fault-tolerance discussed in detail in [14]. For example, simple invocation retry can be efficiently used as recovering action after errors caused by transient network failures (errors 1, 3, 4).

Fig. 6 also provides some information about possibility of differentiating between various errors/failures on the base of

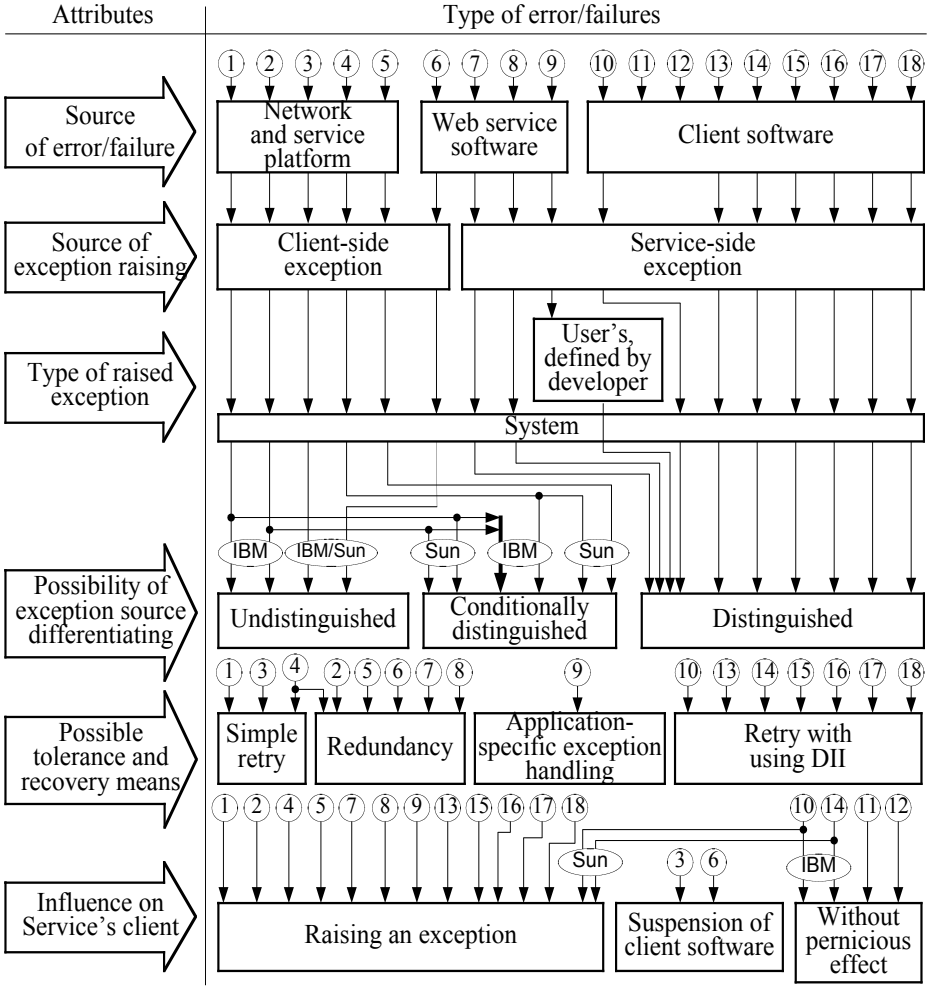

Figure 6. Classification of the SOA-specific errors and failures.

information available from the exceptions description. For example, as it is shown in Fig. 6, we can always differentiate between errors 7 – 18; errors 3 and 6 can not be distinguished from each other independently from development kit used as well as errors 1 and 2 in case of using IBM WSDK.

Finally, other errors can be conditionally distinguished (i.e. distinguished only if a full exception stack trace is available). The numbers in circles in Fig.6 are corresponding to the numbers of SOA-specific errors and failures from Table 1.

6. CONCLUSIONS

Exception handling is widely used as the basis of forward error recovery in service-oriented architecture. Effectiveness of exception handling depends on the features of exceptions raising and on the propagation mechanisms. In our work we have experimented with Web Services, implemented using two development kits: 1) JAX-RPC implementation at Sun Microsystems and 2) IBM WebSphere Software Developer Kit for Web Services. We have performed compatibility analysis of the exception propagation mechanisms and performance implications. This work allows us to draw the following conclusions.

1. Web Services developed by using different toolkits react differently to some DII client errors ("Output parameter type mismatch", "Error in name of input parameter"). Sometimes this diversity can allow us to mask client errors whereas in other cases it will lead to the erroneous outcome.

2. Web Service clients, developed by using different toolkits can have different response time-outs. In our experimentation with simple Web Services we have also observed the starvation of client software developed using the Sun Microsystems toolkit in case of WS hang or packet loss.

3. Not always the exception messages and stack traces gathering in our experimentation were enough to identify the exact cause and, hence, applying an adequate recovery technique. For example, it is not possible to recognize if a remote host is down or it is unreachable because of transient network failures.

4. Web Services developed using different toolkits have different exception propagation time. This affects *failure detection* and *failure notification delay*. It is clear for us that the developers of WSDK should make some effort to reduce this time.

5. Analysis of the exception stack trace and propagation delay can help in identifying of exact sources of the exceptions even if we have caught the sane top-level exception messages. It makes better *fault diagnosis*, which identifies and records the cause of exception in terms of both location and type, and *fault isolation* and *removal*.

6. Knowing the exact cause and sources of exceptions is useful for applying appropriate *failure recovery* or *fault-tolerant means* during exception handling (see Fig. 6). Several types of failures resulting in exceptions can be effectively handled on the client side, whereas others should be handled on the service side. Exceptions handling of the client side errors in early binding procedures may include retry with the help of dynamic invocation. Transient network failures can be tolerated by simple retry. In other cases redundancy and majority voting should be used.

7. Exception statistics gathering and analysis allow improvement of *fault handling* which prevents located faults from being activated again on the base of *system reconfiguration* or *reinitialization*. It is especially related to a composite system with several alternative WS.

8. Analysis of the exception stack trace helps in identifying of the application server, WSDK, libraries and packages used for WS development. This information is useful for *choice of the diverse variants* among the set of alternative Web Services deployed by third parties and building effective fault-tolerant systems using WS redundancy and diversity.

Below we are summarizing our suggestions on how exception handling should be implemented in the SOA systems to help develop systems that handle exceptions optimally. First of all, a Web Service should return exceptions as soon as possible. Long notification delay can significantly affect the SOA performance especially in complex workflow systems. To decrease the exception propagation delay the developers should avoid unnecessary nesting of exceptions and reduce the overall number of exception stack traces. Besides, exceptions should contain more detailed information about cause of error and also provide additional classification attributes to help in error diagnosis and fault tolerance. For example, if an exception reports whether an error seems to be transient or permanent, user's application will be able to automatically choose and perform the most suitable error recovery action (simple retry in case of transient errors or more complicated fault-tolerant techniques otherwise).

In the presented work we have experimented with only two Web Service implementations provided by Sun and IBM. In our future study we are planning to deal with other existing SOA platforms belonging to .NET, AXIS and other categories to have a much wider picture of the exception handling capabilities in SOA.

7. ACKNOWLEDGMENTS

Alexander Romanovsky is supported by the EC ICT DEPLOY project and by the EPSRC/UK TrAmS platform grant.

8. REFERENCES

[1] Chan, Pat. P.W., Lyu, M.R., Malek, M. 2006. Making Services Fault Tolerant. *In D. Penkler, M. Reitenspiess, and F. Tam (Eds.): Service Availability, International Service Availability Symposium, LNCS 4328*, Berlin, Heidelberg: Springer-Verlag, 43–61.

[2] Managing Exceptions in Web Services Environments. 2003. An AmberPoint Whitepaper (http://www.amberpoint.com).

[3] Tartanoglu, F., Issarny, V., Romanovsky, A., Levy, N. 2003. Coordinated Forward Error Recovery for Composite Web Services. In Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS), Florence, Italy, 167-176.

[4] Marsden, E., Fabre, J.-C., Arlat, J. 2002. Dependability of CORBA Systems: Service Characterization by Fault Injection. In Proceedings of the Symposium on Reliable Distributed Systems, Osaka, Japan.

[5] Brambilla, M., Tziviskou, C. 2005. Fundamentals of Exception Handling Within Workflow-Based Web Applications. Journal of Web Engineering (JWE), Vol. 4, Issue 1, 38-56.

[6] Vieira, M., Laranjeiro, N., Madeira, H. 2007. Assessing Robustness of Web-services Infrastructures. In Proceedings of the 2007 Int. Conf. On Dependable Systems and Networks (DSN'2007), 131–136.

[7] Duraes, J., Vieira, M., Madeira, H. 2004. Dependability Benchmarking of Web-Servers. In M. Heisel et al. (Eds.): SAFECOMP 2004, LNCS 3219, 297–310.

[8] Looker, N., Gwynne, B., Xu, J., Munro, M. 2005. An Ontology-Based Approach for Determining the Dependability of Service-Oriented Architectures. In Proceedings of the 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems, USA.

[9] Looker, N., Munro, M., Xu, J. 2005. Simulating Errors in Web Services. International Journal of Simulation Systems, Science & Technology, vol. 5.

[10] Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, Vol. 1, No. 1, 11–33.

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

[12] Gorbenko, A., Mikhaylichenko, A., Kharchenko, V., Romanovsky, A. 2007. Experimenting With Exception Handling Mechanisms Of Web Services Implemented Using Different Development Kits. Technical report CS-TR 1010: http://www.cs.ncl.ac.uk/research/pubs/trs/papers/1010.pdf, Newcastle University

[13] Looker, N., Munro, M., Xu, J. 2004. Testing Web Services. In Proceedings of the 16th IFIP International Conference on Testing of Communicating Systems, Oxford.

[14] Gorbenko, A., Kharchenko, V., Furmanov, A., Tarasyuk, O. 2006. F(I)MEA-Technique of Web Services Analysis and Dependability Ensuring. In M. Butler et al. (Eds.): Rigorous Development of Complex Fault-Tolerant Systems (LNCS 4157), Berlin, Heidelberg: Springer-Verlag, 153–167.

[15] Cristian, F. 1995. Exception Handling and Tolerance of Software Faults. In *Software Fault Tolerance*, M. Lyu, ed., 81-107