Using Inherent Service Redundancy and Diversity to Ensure Web Services Dependability

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

[1] Department of Computer Systems and Networks, National Aerospace University, Kharkiv, Ukraine
A.Gorbenko@csac.khai.edu, V.Kharchenko@khai.edu
[2] School of Computing Science, Newcastle University, Newcastle upon Tyne, UK
Alexander.Romanovsky@newcastle.ac.uk

Abstract. Achieving high dependability of Service-Oriented Architecture (SOA) is crucial for a number of emerging and existing critical domains, such as telecommunication, Grid, e-science, e-business, etc. One of the possible ways to improve this dependability is by employing service redundancy and diversity represented by a number of component web services with the identical or similar functionality at each level of the composite system hierarchy during service composition. Such redundancy can clearly improve web service reliability (trustworthiness) and availability. However to apply this approach we need to solve a number of problems. The paper proposes several solutions for ensuring dependable services composition when using the inherent service redundancy and diversity. We discuss several composition models reflecting different dependability objectives (enhancement of service availability, responsiveness or trustworthiness), invocation strategies of redundant services (sequential or simultaneous) and procedures of responses adjudication.

1 Introduction

The Web Services (WS) architecture [1] based on the SOAP, WSDL and UDDI specifications is rapidly becoming a de facto standard technology for organization of global distributed computing and achieving interoperability between different software applications running on various platforms. It is now extensively used in developing numerous business-critical applications for banking, auctions, Internet shopping, hotel/car/flight/train reservation and booking, e-business, e-science, Grid, etc. That is why analysis and dependability ensuring of this architecture are emerging areas of research and development [1–3]. The WS architecture is in effect a further step in the evolution of the well-known component-based system development with off-the-shelf (OTS) components. The main advances enabling this architecture have been made by the standardisation of the integration process, by a set of interrelated standards such as SOAP, WSDL, UDDI, etc.

Web Services are autonomous systems, the ready-made OTS components belonging to different organizations, without any general or centralised control, that may

M. Butler et al. (Eds.): Fault Tolerance, LNCS 5454, pp. 324–341, 2009.

change their behaviour on the fly. This architecture brings a number of benefits to the users but at the same time poses many challenges to researchers and developers. By their very nature Web Services are black boxes, as neither source code, nor specification, nor information about deployment environment are available; the only known information about them is their interfaces. Moreover, their quality is not completely known and they may not provide sufficient quality of service; it is often safe to treat them as "dirty" boxes, assuming that they always have bugs, do not fit enough, have poor specification and documentation. WSs are heterogeneous, as they might be developed following different standards, fault assumptions, and different conventions and may use different technologies. Finally, their construction and composition are complicated by the fact that the Internet is a poor communication medium (has low quality, not predictable).

The main motivation for our work is the fact that ensuring and assessing dependability of complex service-oriented systems is complicated when these systems are dynamically built or when their components (i.e. Web Services) are dynamically replaced by the new ones with the same (or similar) functionality but unknown dependability characteristics. The lack of evidence about the characteristics of the communication medium, components used in the composition and their possible dependencies makes it extremely difficult to achieve and predict SOA dependability which can vary over a wide range in a random manner. Therefore, users cannot be confident in availability, trustworthiness, reasonable response time and others dependability characteristics. Dealing with such uncertainty, mainly coming from the SOA nature, is the main challenge.

This uncertainty should be treated as the threat (similar and in addition to the commonly known faults, errors and failures). The paper discusses fault-tolerance solutions for building dependable service-oriented systems out of undependable Web Service components, which have changeable functional sets and uncertain dependability characteristics, making use of natural redundancy and diversity inherent to such systems.

In the paper we analyse different dependability-oriented composition models of Web Services and also propose solutions guaranteeing that the overall dependability (availability, correctness and responsiveness) of the composite system is improving.

2 Web Services Redundancy and Diversity

SOA supports construction of the globally distributed massive-scale systems with growing number of services. This makes it unique in allowing access to a number of services with identical or similar functionalities, provided by different vendors and deployed on different platforms all over the Internet. In other words, SOA possesses the inherent redundancy and diversity of the existing Web Services [17]. We should use this fact to build dependable Service-Oriented Systems out of undependable Web Services. Table 1 shows several examples of the existing alternative (redundant) stock quotes and currency exchange Web Services (see [18] for a more detailed discussion of these examples). In [12] the authors present a practical experience report on dependability monitoring of three diverse Bioinformatics Web Services performing

Table 1. An example of alternative (redundant) Web Services

Alternative (redundant) Stock Quotes Web Services
stock_wsx.GetQuote: http://www.webservicex.com/stockquote.asmx?WSDL
stock_gama.GetLatestStockDailyValue: http://www.gama-system.com/webservices/stockquotes.asmx?wsdl
stock_xmethods.getQuote: http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl
stock_sm.GetStockQuotes: http://www.swanandmokashi.com/HomePage/WebServices/StockQuotes.asmx?WSDL
Alternative (redundant) Currency Exchange Web Services
currency_exchange.getRate: http://www.xmethods.net/sd/CurrencyExchangeService.wsdl
currency_convert.ConversionRate: http://www.webservicex.com/CurrencyConvertor.asmx?wsdl

similar BLAST[1] 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, and ii) that they can be used simultaneously to achieve better dependability.

72-87% of the faults in open-source software are independent of the operating environment (i.e. faults in application software) and are hence permanent [20]. Half of the remaining faults are environment depended and permanent. And only 5-14% of the faults are environment depended caused by transient conditions. Hence, software diversity can be an efficient method of fault-tolerance provisioning and decreasing common mode failures caused by software faults [21, 22]. SOA supports inherent diversity at the different level:

1. ***Application diversity:*** i) development diversity of application software (different developers, languages, implementation technologies and tools, etc); ii) data diversity (diversity of data used and data sources); iii) Service diversity (diversity of 'physical' resources and services, for example, flights available, hotel rooms, etc).
2. ***Deployment diversity.*** Different service providers can use diverse deployment environments (different hardware platform, operating systems, web and application servers, DBMS, etc.).
3. ***Spatial (geographical) diversity.*** Redundant Web Services can be dispersed all over the Internet and different service vendors can use different Internet Service Providers.

To build dependable Service-Oriented Systems, developers (systems integrators) and end 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 [23]. Other approach we are discussing in this paper is using all available services simultaneously with the purpose to improve overall system dependability.

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

3 Web Services Dependability

Dependability of a computing system is its ability to timely deliver service that can justifiably be trusted [24]. According to this definition we need to deal with the following dependability attributes, which are relevant to Web Services, and which can be easily measured during WS invocations: (i) availability; (ii) reliability; and (iii) response time (performance). There are several other attributes, describing Quality of Service (QoS), service level agreements (SLAs) and dependability, including authentication, confidentiality, non-repudiation, service cost, etc. [25], 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 service's availability. Availability of a system is a measure of the delivery of correct service with respect to the alternation of correct and incorrect service [24]. It can be defined by a ratio of the system's uptime to all execution time (including downtime). Unfortunately, such technique can be hardly applied for determining the availability of Web Services in a loosely coupled SOA. More adequate, the availability of a Web Service most likely 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). The easier recovery action here is simple retry.

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 complicated, taking into account the fact that service invocation rate can vary in a wide range for different services and services customers. Another problem here is that Web Service returns errors of two main types [9]:

1. Evident erroneous response which results in exception message. The probability of such errors can be measured by the proportion of the total number of service invocation number of exception messages received (apart from exception "HTTP Status-Code (404): Not Found" that indicate about service unavailability). If such error occurs, user could retry the same service latter or (most likely) invoke an alternative one.
2. Non-evident erroneous response. It can be present in a 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 unpredicted consequences, and, as a result, service discredit. Detection of such errors is possible by comparing service response with response from another diverse service.

Therefore, the key problem services developers and users are faced with is enhancing the service trustworthiness (correctness) rather then decreasing probability of exception (i.e. evident error occurrence).

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 waits 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 be hardly predicted due to the uncertain network fluctuations whereas connection waiting time and execution time depend on service load and throughput.

4 Web Services Composition

Web service composition is currently an active area of research, with many languages being proposed by academic and industrial research groups. IBM Web Service Flow Language (WSFL) [4] and Microsoft's XLANG [5] were two of the earliest languages to define standards for Web services composition. Both languages extended W3C Web Service Description Language (WSDL) [6], which is the standard language for describing the syntactic aspects of a Web service. Business Process Execution Language for Web Services (BPEL4WS) [7] is a recently proposed specification that represents the merging of WSFL and XLANG. BPEL4WS combines the graph oriented process representation of WSFL and the structural construct based processes of XLANG into a unified standard for Web services composition.

In addition to these commercial XML-based standards, there have been work on a unique Web service composition language called Web Ontology Language for Services (www.daml.org/services) OWL-S (previously known as DAML-S) [8], which provides a richer description of Web service compositions by specifying its semantics.

In our work we focus on the general patterns (types) of the WS composition and identify two typical blueprints of composing WSs: i) "vertical" composition for functionality extension, and ii) "horizontal" composition for dependability improvement (dependability-oriented composition).

The first type of service composition ("vertical") is used for building the WorkFlow (WF) of the systems and is already supported by BPEL, BPML, XPDL, JPDL and other WF languages. The second one ("horizontal") deals with a set of redundant (and possibly diverse) Web Services with identical or similar functionality. Rather than investigate fixed redundancy schemes [28] (like two-out-of-three or three-out-of-five) in this paper we discuss flexible patterns improving various dependability attributes (availability, trustworthiness or responsibility) taken separately.

Bellow we show some illustrative examples of the two types of WSs composition and discuss the way in which the *horizontal* composition improves dependability of SOA. We use the web-based *Travel Agency* as an example of Web Services composition, this example has been extensively used by other researchers [3, 26, 27].

4.1 *Vertical* Composition for Functionality Extension

The "vertical" composition (Fig. 1-a) extends the Web Services functionality. A new Composite Web Service is composed out of several particular services which provide different functions. For example, the *Travel Agency* (TA) Service can be composed of a number of services such as *Flight Service*, *Car Rental Service*, *Hotel Service*, etc.

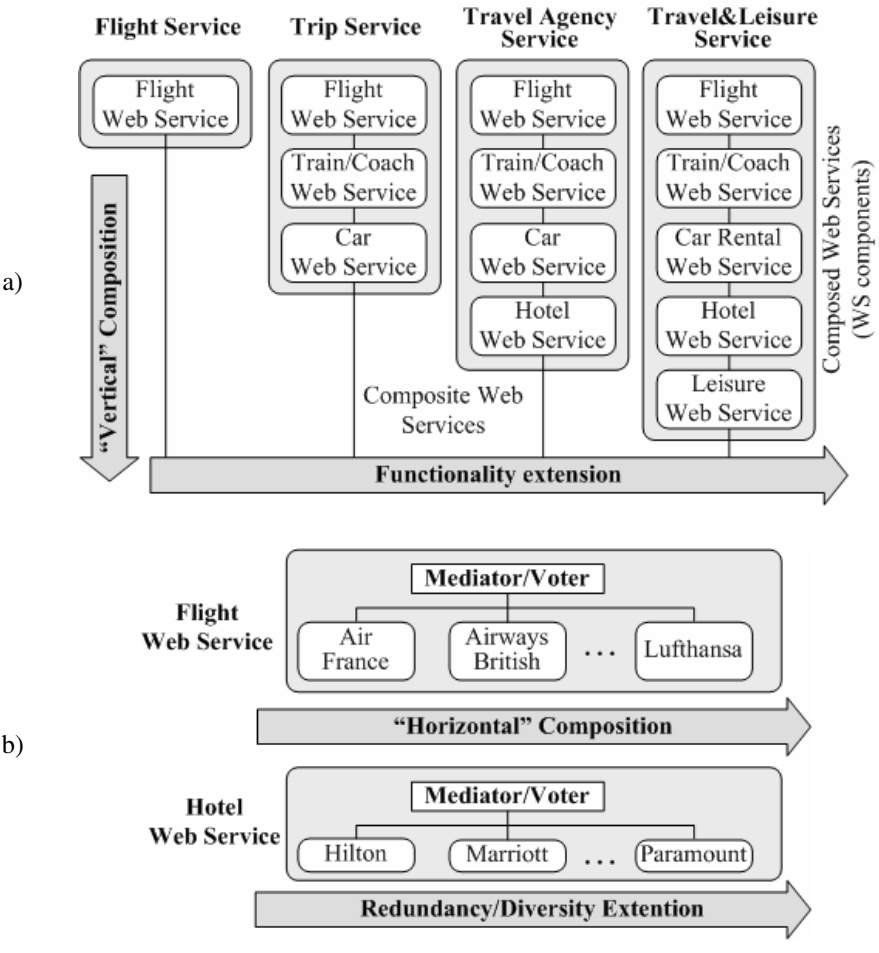


Fig. 1. Web Services Composition: a) "vertical"; b) "horizontal"

The main invocation parameters for such a composite *TA Service* are trip endpoint (country and city), dates (and time) of arrival and departure, user details and preferences. The *TA Service* then invokes corresponding services which books flight/train/coach tickets, hotel room, rents a car, etc.

The Composite Web Service can invoke a set of target (composed) services simultaneously to reduce the mean execution time or sequentially (if execution of one service depends on the result of another one).

If some of the services fail or cannot satisfy the user's request (for example, when there are no flights available for specified dates) all other services have to be rolled back and their results should be cancelled.

To improve dependability of such composite system various means of fault-tolerance and error recovery should be applied, including redundancy, exception handling, forward error recovery, etc.

4.2 *Horizontal* Composition for Dependability Improvement

The "horizontal" composition (Fig. 1-b) uses several alternative (diverse) Web Services with the identical or similar functionality, or several operational releases of the same service. Such kind of redundancy based on inherent service diversity improves service availability and reliability (correctness and trustworthiness) of Web Service composition.

Architecture with the "horizontal" (dependability-oriented) composition includes a "Mediator" component, which adjudicates the responses from all diverse Web Services and returns an adjudicated response to the consumer. In the simplest case the "Mediator" is a *voter* (i.e. performs majority voting using the responses from redundant Web Services). It can be also programmed to perform more complex operation like aggregation, or provide the best choice according to selection criterions specified by user (for example, highest possible exchange rate or minimal asked quotation).

Papers [10-13] introduce special components (called "Service Resolver", "Proxy", "Service Container" or "Wrapper") with the similar functionality.

5 Middleware-Based Architecture Supporting Dependability-Oriented Composition

5.1 Patterns of Dependability-Oriented Composition

In [9] we proposed an architecture which uses a dedicated middleware for a managed dependable upgrade and the "horizontal" composition of Web Services. The middleware runs several redundant (diverse) Web Services (Fig. 2). It intercepts the user's requests coming through the WS interface, relays them to all the redundant services and collects the responses. It is also responsible for dependability measurement and publishing the confidence in dependability [9] associated with each service.

The architecture proposed supports several composition models meeting different dependability objectives (such as enhancement of service availability, responsiveness or trustworthiness), various strategies for invoking redundant services (sequential or simultaneous) and procedures for response adjudication. These models form *patterns* of dependability-oriented composition. The basic ones are:

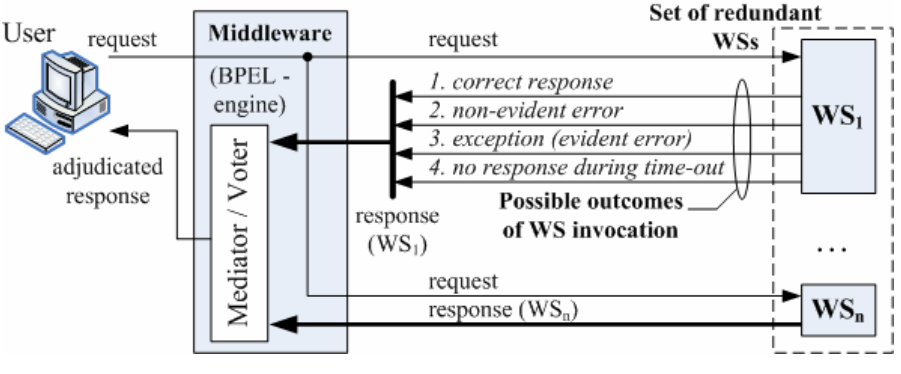
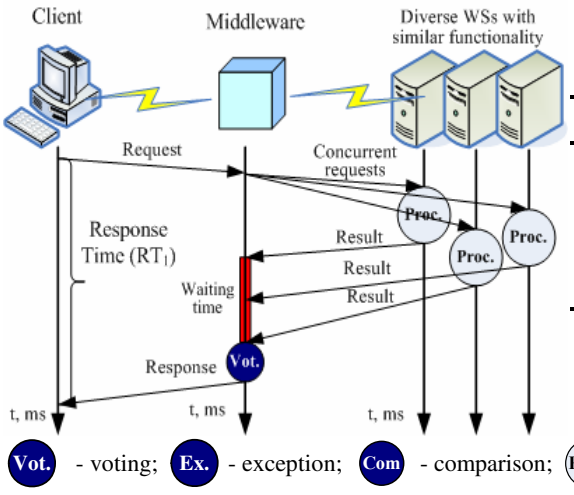


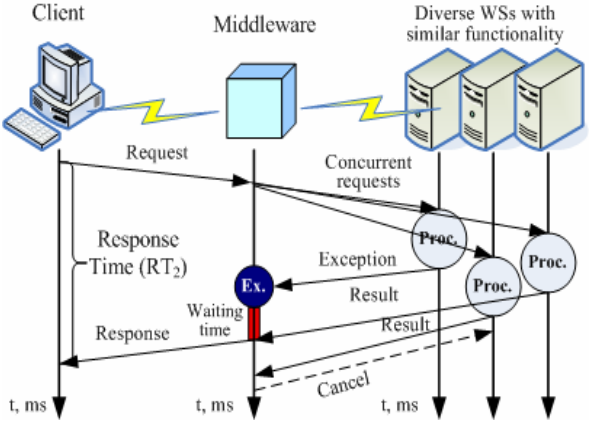
Fig. 2. Architecture of dependability-oriented composition



Simulation results:

	Case1	Case2	Case3
P(cr)	0.966	0.936	0.928
P(ne)	0.010	0.048	0.013
P(ex)	0.016	0.008	0.032
P(ua)	0.008	0.008	0.027
MRT	397.60	397.60	431.40

Fig. 3. Simulation results of the *Reliable concurrent execution* pattern



Simulation results:

	Case1	Case2	Case3
P(cr)	0.961	0.915	0.922
P(ne)	0.003	0.017	0.005
P(ex)	0.029	0.060	0.046
P(ua)	0.008	0.008	0.027
MRT	141.10	133.85	163.82

Fig. 4. Simulation results of the *Fast concurrent execution* pattern

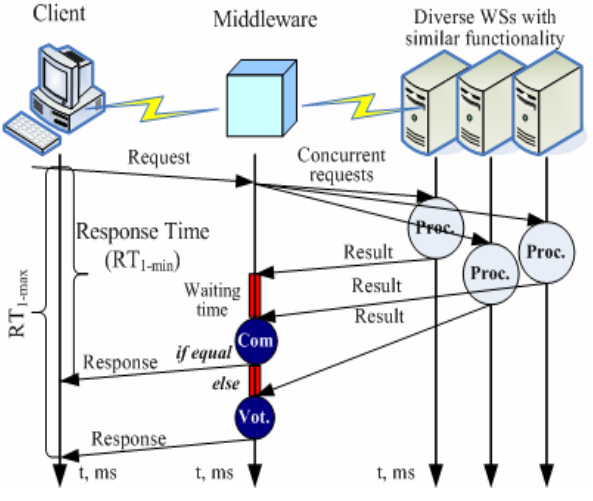
1. *Reliable concurrent execution* for trustworthiness improvement (Fig. 3). All available redundant (diverse) WSs are invoked concurrently and their responses are used by the middleware to produce an adjudicated response to the consumer of the WS (i.e. voting procedure). Initial values of measures *P(cr), P(ne), P(ua), P(ex)* and mean response time (MRT) used in three different simulation cases are discussed in section 5.2.
2. *Fast concurrent execution* for responsiveness improvement (Fig. 4). All available redundant (diverse) WSs are invoked concurrently and the fastest non-evidently incorrect response is returned to the service consumer.
3. *Adaptive concurrent execution* (Fig. 5). All (or some of) redundant (diverse) WSs are executed concurrently. The middleware is configured to wait for up to a certain

number of responses to be collected from the redundant services, but no longer than a pre-defined timeout.

4. *Sequential execution* for minimal service loading (Fig. 6). The subsequent redundant WS is only invoked if the response received from the previous one is evidently incorrect (i.e. exception).

5.2 Simulation

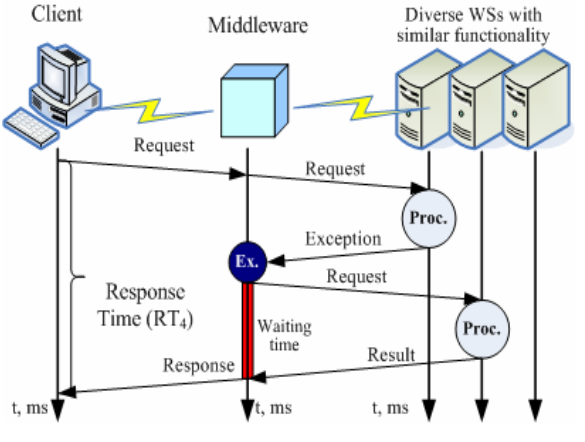
Effectiveness of the different composition models depends on the probability of service unavailability, occurrence of evident (exceptions raised) and non-evident (erroneous results returned) failures. The probability of service unavailability due to



Simulation results:

	Case1	Case2	Case3
P(cr)	0.966	0.936	0.928
P(ne)	0.010	0.048	0.013
P(ex)	0.016	0.008	0.032
P(ua)	0.008	0.008	0.027
MRT	289.72	289.08	330.52

Fig. 5. Simulation results of the *Adaptive concurrent execution* pattern



Simulation results:

	Case1	Case2	Case3
P(cr)	0.961	0.915	0.922
P(ne)	0.003	0.017	0.005
P(ex)	0.029	0.060	0.046
P(ua)	0.008	0.008	0.027
MRT	283.99	266.25	387.86

Fig. 6. Simulation results of the *Sequential execution* pattern

different reasons (service overload, network failures, and congestions) is several orders greater than probability of failure occurrence. Moreover, different exceptions arise during service invocation more frequently than non-evident failures occur.

To analyse the effectiveness of the proposed patterns we developed a simulation model running in the MATLAB 6.0 environment. It used the following initial values chosen using the real-life statistics [19]:

	Case 1	*Case 2*	*Case 3*
P(correct response), P(cr)	*0.70*	*0.70*	*0.60*
P(non-evident error), P(ne)	*0.01*	*0.05*	*0.01*
P(exception), P(ex)	*0.09*	*0.05*	*0.09*
P(service unavailability), P(ua)	*0.20*	*0.20*	*0.30*

Each redundant Web Service was modelled as a black box that is assumed to fail independently of all the others but with the same probability [26]. The first two cases (cases 1 and 2) correspond to services that have the same availability but different probabilities of evident and non-evident error occurrence (we assume more trusted services in case 1). The third one simulates trusted service with worse availability as compared to Case 1 (possibly, due to narrow network bandwidth or frequent congestions). During simulation we also set *Mean Response Time* (MRT) which equals 200 ms and *Maximum Waiting Time* (time-out) which equals 500 ms.

The simulation results of each proposed patterns of dependability-oriented composition are shown at the Figures 3 – 6 respectively. Figure 7 gives a summary of all simulation cases.

A practical application of the horizontal composition requires developing new workflow patterns and languages constructs, supporting different composition models and procedures of multiple results resolving and voting.

6 Implementation

6.1 Work-Flow Patterns Supporting Web Services Composition

The workflow patterns capture typical control flow dependencies encountered during workflow modelling. There are more then 20 typical patterns used for description of different workflow constructions of "vertical" composition [14]. The basic ones are: *'Sequence'*, *'Exclusive Choice'*, *'Simple Merge'*, *'Parallel Split'*, *'Synchronization'*, *'Discriminator'*, *'Regular Cycle'*, etc.

Each WF language describes a set of elements (activities) used for implementing different WF patterns. For example, BPEL4WS defines both primitive (*'invoke'*, *'receive'*, *'reply'*, *'wait'*, *'assign'*, *'throw'*, *'terminate'*, *'empty'*) and structured (*'sequence'*, *'switch'*, *'while'*, *'flow'*, *'pick'*, *'scope'*) activities.

The first ones are used for intercommunication and invoking operations on some web service. Structured activities present of complex workflow structures and can be nested and combined in arbitrary ways.

		P(cr)	P(ne)	P(ex)	P(ua)	MRT, ms
Case 1	WS_i	0.7	0.01	0.09	0.20	200
	Pattern1	0.966	0.01	0.016	0.008	397.6
	Pattern2	0.961	0.003	0.029	0.008	141.1
	Pattern3	0.966	0.01	0.016	0.008	289.72
	Pattern4	0.961	0.003	0.029	0.008	283.99
Case 2	WS_i	0.7	0.05	0.05	0.20	200
	Pattern1	0.936	0.048	0.008	0.008	397.6
	Pattern2	0.915	0.017	0.06	0.008	133.85
	Pattern3	0.936	0.048	0.008	0.008	289.08
	Pattern4	0.915	0.017	0.06	0.008	266.25
Case 3	WS_i	0.6	0.01	0.09	0.30	200
	Pattern1	0.928	0.013	0.032	0.027	431.4
	Pattern2	0.922	0.005	0.046	0.027	163.82
	Pattern3	0.928	0.013	0.032	0.027	330.52
	Pattern4	0.922	0.005	0.046	0.027	387.86

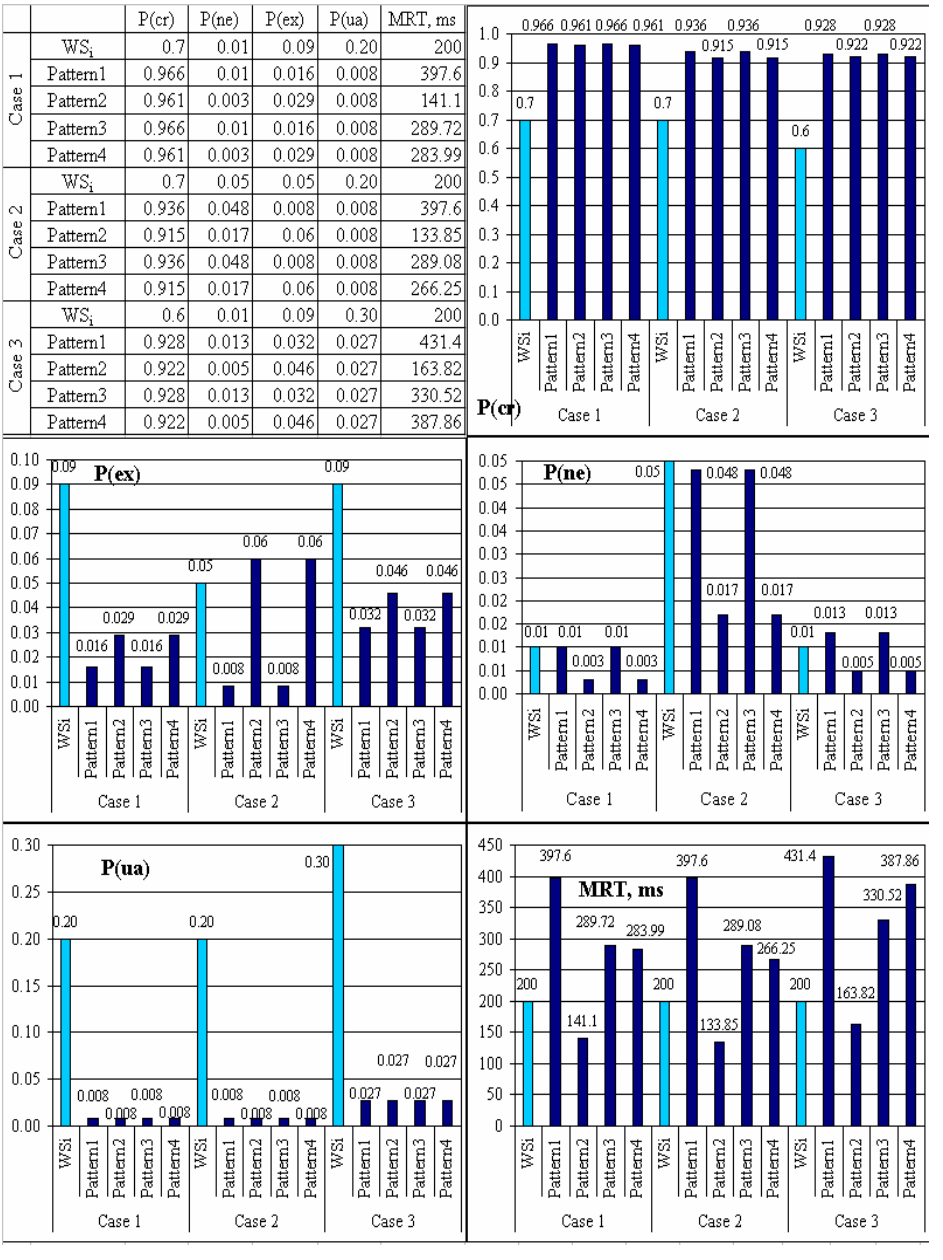


Fig. 7. Summary of simulation results (WSi – particular Web Service; Pattern 1 – the *Reliable concurrent execution* pattern; Pattern 2 – the *Fast concurrent execution* pattern; Pattern 3 – the *Adaptive concurrent execution* pattern; Pattern 4 – Simulation results of the *Sequential execution* pattern)

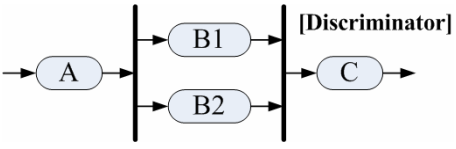


Fig. 8. Workflow pattern *Discriminator*

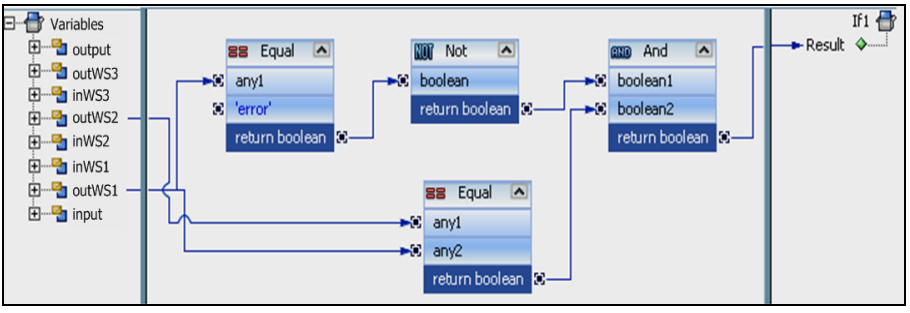
In fact, the only one of the basic WF patterns *Discriminator* fits for implementing the *Fast Concurrent Execution* pattern providing maximum responsiveness. Discriminator (see Fig. 8) is a point in the workflow process that waits for one of the incoming branches to complete before activating the subsequent activity. The first one that comes up with the result should proceed the workflow. The other results will be ignored.

However, only BPML `<all>` and BPEL `<pick>` activities support such WF pattern [15, 16] (Fig. 9). To support dependability-oriented composition the additional WF patterns need to be developed and implemented for different WF languages.

The new activities allowing a business process to support *redundancy* and perform *voting* procedure should also be developed. This is a motivation of our further work.

```
<process name="PatternDiscriminator">
  <sequence> <context>
      <signal name="completed_B"/>
      <process name="B1">
         …
        <raise signal="completed_B"/>
      </process>
      <process name="B2">
         …
        <raise signal=" completed_B"/>
      </process>
    </context>
    <action name="A" …>
      …
    </action>
    <all>
      <spawn process="B1"/>
      <spawn process="B2"/>
    </all>
    <synch signal="completed_B"/>
    <action name="C" …>
      …
    </action>
  </sequence> </process>
```

Fig. 9. BPML implementation of the *Discriminator* pattern




```
<if name="If1">
    <condition> ( not( ( $outWS1 = 'error' ) )
     and ( $outWS1 = $outWS2 ) ) </condition>
    <sequence name="Sequence4">
       <assign name="Assign2">
          <copy>
             <from variable="outWS1"/>
             <to variable="output"/>
          </copy>
       </assign>
    </sequence>
    <else>
       <sequence name="Sequence4">
          <invoke name="Invoke3" partnerLink="WS3" .../>
  ...
```

Fig. 10. WS-BPEL *If* statement implementing responses matching (graphic and text notion)

6.2 Testbed Workflows

The patterns of dependability-oriented composition discussed above have been implemented as a set of testbed workflows (see, for example, Fig. 11) by using a graphics-based BpelModule which is a part of IDE NetBeans 6.0[2].

We used WS-BPEL 2.0[3] specification, which introduces two constructs specifically for extensions ('*extensionActivity*' and '*extensionAssignActivity*'). It also has improved fault handling and process termination features.

New fault handlers having '*catch*', '*catchAll*', '*compensate*', '*throw*' and '*rethrow*' constructs as well as '*terminationHandler*' and '*exitOnStandardFault*' activities were added in WS-BPEL.

A business logic supporting voting and comparison procedures within particular pattern is implementing by using *if* statements.

Fig. 10 gives an example of how to compare the responses from first two services invoked at the first step of the *Adaptive concurrent execution* pattern (Fig. 5 and 11). If the results are equal and are not 'exception' (standard error) then the agreed result can be returned to the user, otherwise the third service should be invoked to perform voting procedure.

[2] www.netbeans.org/community/releases/60/
[3] www.oasis-open.org/committees/wsbpel/

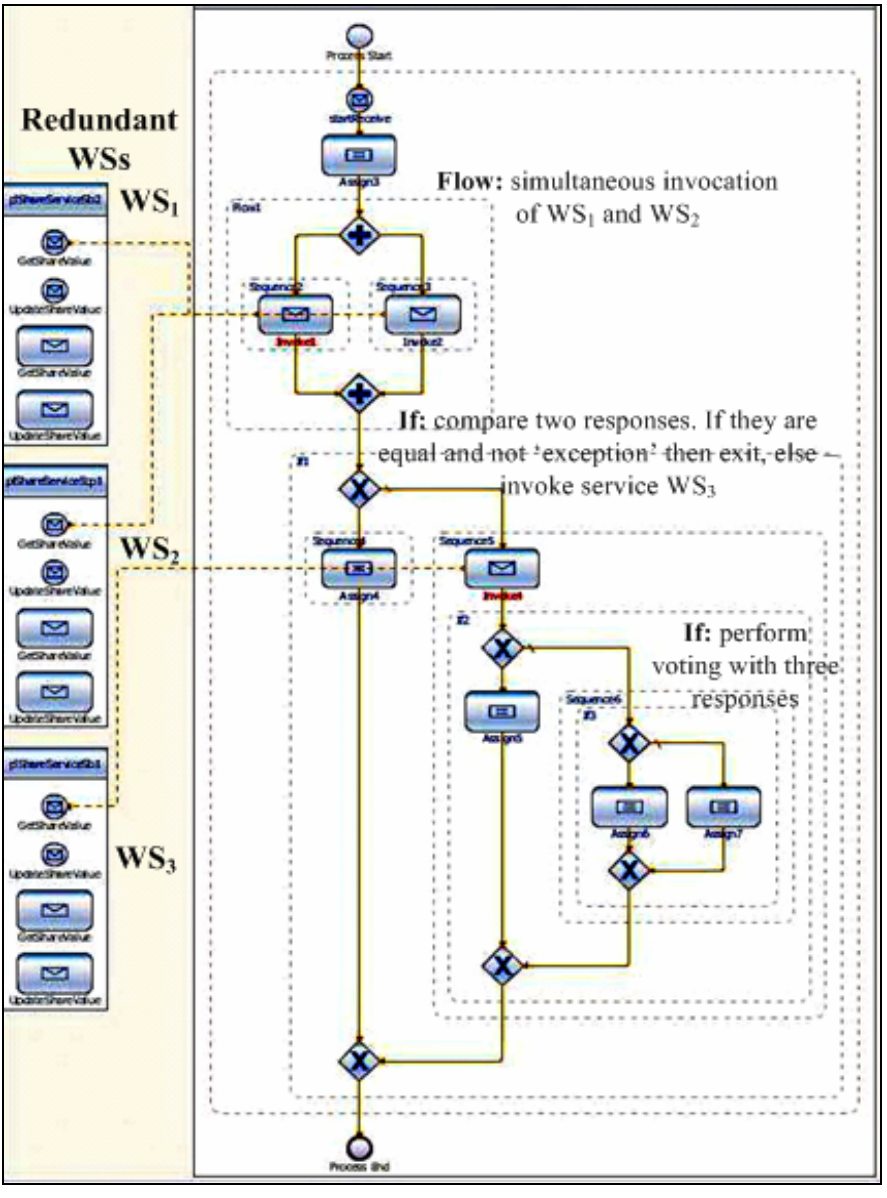


Fig. 11. Screen shot of WS-BPEL-workflow implementing the *Adaptive concurrent execution* pattern

7 Conclusions and Future Researches

We have addressed different models of a Web Services composition which extend functionality ("vertical" composition) or improve dependability ("horizontal" composition).

"Vertical" composition uses redundancy based on natural diversity of existing Web Services with the identical or similar functionality deployed by third parties. We discussed middleware-based architecture that provides dependability-oriented composition of Web Services. For the best result middleware has to implement on-line monitoring and dependability control.

"Horizontal" composition, which uses redundancy in combination with diversity, is one of the promising means of enhancing service availability and providing fault-tolerance. Different patterns are applicable here. As it is shown from simulation and experimentation, all models of dependability-oriented composition significantly improve service availability (as it was expected) and probability of correct response (see Table 2).

The *Adaptive concurrent execution* and *Reliable concurrent execution* patterns give maximal reliability (probability of correct response) and minimal probability of exception at the expense of performance deterioration. However the *Adaptive concurrent execution* pattern provides better reliability-to-response-time ratio.

The *Fast concurrent execution* and *Sequential execution* patterns improve service correctness (decrease probability of non-evident error). Besides, the first one provides minimal response time (less than mean response time of each particular WS). An unexpected result was that the *Sequential execution* pattern improves reliability and correctness without performing unnecessary services invocation and, at the same time, provides rather good response time.

Finally, a more complex composition model combining the "vertical" and "horizontal" compositions is also possible. It supports two boundary architectures:

1. *Multilevel mediation* (Fig. 12-a) with co-ordination at each level of functional composition.
2. *One-level mediation* (Fig. 12-b) with co-ordination at only the top level.

A number of intermediate architectures are also possible. However, questions like "How many horizontal composition levels will provide the maximal improvement?", "When mediator (voter) should be placed?" are yet unsolved and are objectives of future researches.

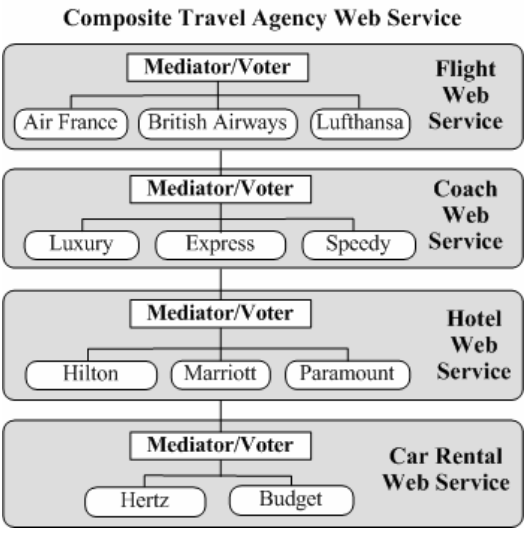
Another question is how to assess and take into account actual services diversity. Because it is obvious that different Web Services can refer to the same 'physical' services or resources like it is shown in the Fig, 12-b where two independent *TA Services* (v.1 and v.2) use the same *Car Rental Service* 'Hertz'.

Table 2. Effectiveness of different patterns of dependability-oriented composition

№	Patterns of dependability-oriented composition	*max. probability of correct response*	*min. probability of non-evident error*	*min. probability of exception*	*min. probability of service unavailability*	*min. response time*
1	Reliable concurrent execution	++		+	+	--
2	**Fast concurrent execution**	**+**	**+**		**+**	**+**
3	Adaptive concurrent execution	+	+		+	-
4	**Sequential execution**	**++**		**+**	**+**	-

('+' – *advantage;* '-' – *disadvantage*)

a)



b)

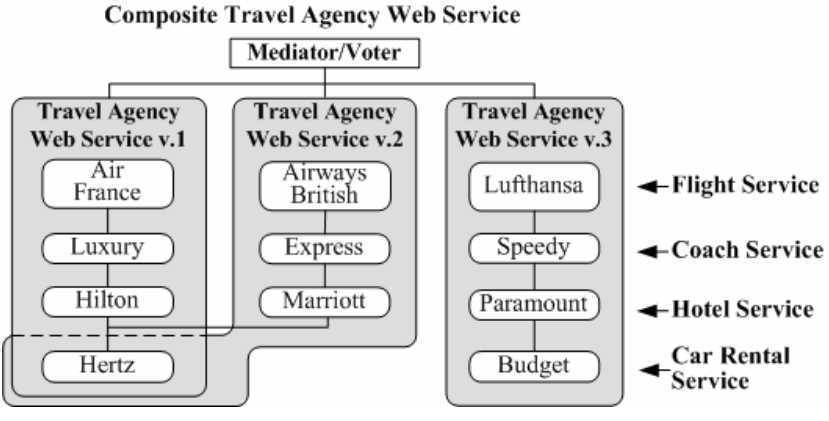


Fig. 12. Combined "vertical and horizontal" Web Service composition using *multilevel* (a) and *one-level* (b) mediation

Applying in practice techniques of the "horizontal" composition and other means of improving SOA dependability requires developing new workflow patterns and implementing them in different WF languages.

In our future work we are going to use WS-BPEL 2.0, which is a new version of popular language widely-used in industry for the specification of business processes and business interaction protocols. Of a particular interest to us is its support for extensibility by allowing namespace-qualified attributes to appear in any standard element and by allowing new user-specified activities to be defined by using '*extensionActivity*' and '*extensionAssignActivity*' constructions.

Acknowledgments. Alexander Romanovsky is partially supported by the EC ICT DEPLOY project.

References

1. W3C Working Group. Web Services Architecture (2004), http://www.w3.org/TR/ws-arch/
2. Ferguson, D.F., Storey, T., Lovering, B., Shewchuk, J.: Secure, Reliable, Transacted Web Services: Architecture and Composition. Microsoft and IBM Technical Report (2003), http://www-106.ibm.com/developerworks/webservices/library/ws-securtrans
3. Tartanoglu, F., Issarny, V., Romanovsky, A., Levy, N.: Dependability in the Web Service Architecture. In: Architecting Dependable Systems, pp. 89–108. Springer, Heidelberg (2003)
4. Leymann, F.: Web Services Flow Language. Technical report, IBM (2001)
5. Thatte, S.: XLANG: Web Services for Business Process Design. Technical report, Microsoft (2001)
6. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: WSDL: Web services description language (2001), http://www.w3.org/TR/wsdl
7. Andrews, T., Cubera, F., Dholakia, H.: Business Process Execution Language for Web Services Version 1.1. OASIS (2003), http://ifr.sap.com/bpel4ws
8. Ankolekar, et al.: Ontology Web Language for Services (OWL-S) (2002), http://www.daml.org/services
9. Gorbenko, A., Kharchenko, V., Popov, P., Romanovsky, A.: Dependable composite web services with components upgraded online. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems III. LNCS, vol. 3549, pp. 92–121. Springer, Heidelberg (2005)
10. Hall, S., Dobson, G., Sommerville, I.: A Container-Based Approach to Fault Tolerance in Service-Oriented Architectures (2005), http://digs.sourceforge.net/papers/2005-icse-paper.pdf
11. Maheshwari, P., Erradi, A.: Architectural Styles for Reliable and Manageable Web Services (2005), http://mercury.it.swin.edu.au/ctg/AWSA05/Papers/erradi.pdf
12. Chen, Y., Romanovsky, A., Li, P.: Web Services Dependability and Performance Monitoring. In: Proc. 21st Annual UK Performance Engineering Workshop, UKPEW 2005 (2005), http://www.staff.ncl.ac.uk/nigel.thomas/UKPEW2005/ukpew-proceedings.pdf
13. Townend, P., Groth, P., Xu, J.: A Provenance-Aware Weighted Fault Tolerance Scheme for Service-Based Applications. In: Proc. of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (2005)
14. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Pattern-Based Analysis of BPEL4WS. QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, Australia (2002), http://is.tm.tue.nl/staff/wvdaalst/publications/p175.pdf
15. van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M., Wohed, P.: Pattern-Based Analysis of BPML (and WSCI). QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, Australia (2002), http://is.tm.tue.nl/research/patterns/download/qut_bpml_rep.pdf

16. Wohed, P., van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Analysis of web services composition languages: The case of BPEL4WS. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) ER 2003. LNCS, vol. 2813, pp. 200–215. Springer, Heidelberg (2003)
17. Gorbenko, A., Kharchenko, V., Romanovsky, A.: Vertical and Horizontal Composition in Service-Oriented Architecture. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 139–147. Springer, Heidelberg (2007)
18. Ernst, M.D., Lencevicius, R., Perkins, J.H.: Detection of Web Service substitutability and composability. In: Proc. International Workshop on Web Services Modeling and Testing (WS-MaTe 2006), pp. 123–135 (2006)
19. Chen, Y., Romanovsky, A.: Improving the Dependability of Web Services Integration. IT Professional: Technology Solutions for the Enterprise. IEEE Computer Society, issue, pp. 20–26 (January/February 2008)
20. Chandra, S., Chen, P.M.: Whither Generic Recovery From Application Faults? A Fault Study using Open-Source Software. In: Proc. Int. Conf. on Dependable Systems and Networks, pp. 97–106 (2000)
21. Deswarte, Y., Kanoun, K., Laprie, J.-C.: Diversity against Accidental and Deliberate Faults Computer Security. In: Dependability and Assurance: From Needs to Solutions. IEEE Computer Society Press, Washington (1998)
22. Lyu, M.R. (ed.): Handbook of Software Reliability Engineering, 805 p. McGraw-Hill Company, New York (1996)
23. Wang, Y., Vassileva, J.: Toward Trust and Reputation Based Web Service Selection: A Survey. In: Proc. International Transactions on Systems Science and Applications (ITSSA) Journal, special Issue on New tendencies on Web Services and Multi-agent Systems (WS-MAS), vol 3(2) (2007)
24. Avizienis, J.-C., Laprie, B., Randell, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (2004)
25. Yang, S., Lan, B., Chung, J.-Y.: Analysis of QoS Aware Web Services. In: Proc. International Computer Symposium on Web Technologies and Information Security Workshop (ICS) (2006)
26. Kaâniche, M., Kanoun, K., Martinello, M.: A User-Perceived Availability Evaluation of a Web Based Travel Agency. In: Proc. International Conference on Dependable Systems and Networks (DSN 2003), pp. 709–718 (2003)
27. Thomas, A., Venter, L.: Propagating Trust In The Web Services Framework. In: Proc. Information Security South Africa Conference (ISSA 2004), http://icsa.cs.up.ac.za/issa/2004/Proceedings/Full/012.pdf
28. Pat, P.W., Chan, M., Lyu, R., Malek, M.: Making Services Fault Tolerant. In: Proc. 3rd International Service Availability Symposium (ISAS 2006) (2006), http://www.cse.cuhk.edu.hk/~lyu/paper_pdf/ISAS06.pdf