Reliability Modeling for SOA Systems

G. Delac, M. Silic and S. Srbljic
University of Zagreb Faculty of Electrical Engineering and Computing, Croatia
{goran.delac, marin.silic, sinisa.srbljic}@fer.hr

Abstract - Service-oriented architecture (SOA) is a popular paradigm for development of distributed systems by composing the functionality provided by the services exposed on the network. In effect, the services can use functionalities of other services to accomplish their own goals. Although such an architecture provides an elegant solution to simple construction of loosely coupled distributed systems, it also introduces additional concerns. One of the primary concerns in designing a SOA system is the overall system reliability. Since the building blocks are services provided by various third parties, it is often not possible to apply the well established fault removal techniques during the development phases. Therefore, in order to reach desirable system reliability for SOA systems, the focus shifts towards fault prediction and fault tolerance techniques. In this paper an overview of existing reliability modeling techniques for SOA-based systems is given. Furthermore, we present a model for reliability estimation of a service composition using directed acyclic graphs. The model is applied to the service composition based on the orchestration model. A case study for the proposed model is presented by analyzing a simple Web Service composition scenario.

I. INTRODUCTION

Service-oriented architecture provides guidelines for development of loosely coupled distributed systems. At its core are services, applications deployed on various network nodes accessible through a unified interface. The most common implementation of SOA is the *Web Services* technology (*WS*) [8]. However, other approaches in implementing SOA are also possible, like utilizing the *REST* architectural style. One of the greatest advantages of SOA is the ability to combine several services and redeploy them as a new service with added-value, i.e. service composition. This makes SOA particularly popular in designing enterprise systems, since different parts of business process functionalities can be deployed by various organizational units. Furthermore, the service composite behavior is easily modified and thus the SOA-based systems provide great flexibility in environments where demands constantly change. However, designing service compositions also presents additional challenges since the services can be deployed by third parties over which the composition designer has no control. A possible concern in such an environment is the necessity to design a composition with a specific level of reliability.

In this paper we introduce a software reliability growth model for service composition development. The basis for reasoning about service composition reliability is the knowledge of reliability for all atomic services that are used as composition's building elements. We define a model used to estimate the reliability of atomic services by utilizing a testing framework. Furthermore, we propose to model the reliability of a service composition by using directed acyclic graphs. In addition, a simple method for detecting weak points in the composition reliability model is introduced. We propose to strengthen the weak points by utilizing fault tolerance methods. Finally, we give an overview of the existing methods for service reliability estimation.

The rest of the paper is structured as follows: In Section 2, an overview of the existing methods used to calculate reliability of SOA-based systems is presented. Service composition development model aimed at improving the composition reliability is presented in Section 3. The model is complemented by a service composition use case example. Section 4 concludes the paper with discussion on model's accuracy.

II. RELATED WORK

In [1] authors present a model used to estimate the reliability of the Web services platform. A general framework for testing and reliability assessment for Web services and its integration into the Web service architecture is presented. The proposed reliability model takes into account both the reliability of an individual (atomic) Web service as well as the overall reliability of service compositions. In order to avoid the usage of monitors for fault detection in atomic services, a voting method is used. The applied approach is similar to N-version programming [10]. Each atomic service has to be assigned a set of services with equivalent functionality. The faults are detected by applying the majority vote principle, i.e. the fault is detected if consensus cannot be reached. The presented model for atomic services calculates the reliability in regular time intervals based on the vote outcomes in each interval. A clear advantage of this approach is the ability to measure the reliability during system's operation. However such an approach implies the usage of multiple variants or replicas of the same service in order to estimate reliability, which is resource demanding. The reliability values of atomic services are used to compute the overall service composition reliability. Specifically, the model estimates the reliability of a scenario defined by a set of operations performed on the atomic services. Each scenario is weighted by its execution rate and can consist of operations executed in a sequence, concurrently or repetitively, i.e. in a loop.

The approach considered in [2] calculates the overall quality of service of an individual service in a service composition. The estimated quality is to serve as guidance for the developers to detect weak points in their compositions. For each service a quality matrix is

maintained which can hold multiple quality parameters. For reliability purposes authors suggest monitoring availability and accessibility. Also, an involvement matrix is used to define how relevant the functionalities of a particular service are in a service composition. By utilizing these two matrices weight factors for each service can be calculated. The weight factors are multiplied by service quality parameters to get the overall quality of service. The authors propose a monitoring environment in which the services are accessed through a unified mediator. The mediator performs quality measurements and stores the results in the repository. The analyzer then extracts the measurements from the repository and performs the reliability estimations.

In [3] the authors propose to add fault tolerance support for service-oriented systems by extending the SOA meta model. The model is extended with additional components that provide means to model fault detection and mitigation processes. The basic components used are: monitor, comparator, selector and policy. When a fault is detected, monitors are placed in the model to localize the faulty behavior to a specific component. Then, the component is replaced by the selector with a functionally equivalent component that displays correct behavior. To detect the functionally equivalent component, a comparator is used. The comparator scans service descriptions available in the service registry and groups the services according to their similarity. Finally the policy component holds all the policies for the selector component which can take into account time, cost and dependability parameters. The main disadvantage of this model is dependence on similar services created by the comparator component. The service descriptions often do not contain enough useful information to properly group services. In addition, although proper monitor placement is crucial for the model's performance, it is not addressed by the authors.

A framework for selection of optimal fault tolerance strategies for Web services is presented in [4]. The authors suggest harnessing time-redundancy and space-redundancy strategies as well as their combinations to facilitate fault tolerance for Web services. A model used to calculate failure rate and response time based on the values for individual services is presented. The model is implemented as a distributed evaluation framework. The framework should be run by an independent third party and its main purpose is to relieve the users of the necessity to measure performance of Web services. With the measured performance parameters an evaluation can be performed for each fault tolerance strategy. The results are further processed by a utility function, matching them with the user requirements. Based on the calculated utility function, the optimal fault strategy for a particular service and user requirements is suggested. In order to cope with the varying performance of web services, a dynamic reconfiguration approach is presented in which service performance parameters and user requirements are frequently updated to recalculate the optimal strategy. However, although frequent testing can have an impact on the service performance itself, it is not addressed by this paper. The authors outline that the approach is valid only for stateless services and that the quality of service parameters could be extended.

In [5] the author proposes an approach based on service level replication, i.e. space-redundancy method, to construct a fault tolerance framework for web services. The approach is founded on the consensus-based replication algorithm inspired by Paxos protocol family. During each client request one replicated service takes the role of primary replica. The primary replica communicates with other replicas attempting to reach the consensus on the given client request. If the request is supported by the majority of replicas, the primary replica returns the response to the client node. During the next request, another replica can take the role of the primary replica. The presented approach is lightweight, since it can be easily integrated with the existing WS infrastructures. Although no additional infrastructure, such as monitors or mediators is required, replicated services need to be modified to implement the consensus protocol.

In [6] the author presents a methodology to select an optimal web service composition based by QoS and risk parameters. If more than one viable service composition is possible, the optimal one is selected in two steps. First, the composition with the best QoS parameters is selected, i.e. the composition in which each atomic service performs the best. Second, the composition plan is described with a Petri net and then translated into a fault tree used to calculate the overall composition risk. Finally the composition is modified by selecting appropriate services on basis of finding an optimal balance between the QoS and the overall composition risk. The author suggests performing the optimization by linear programming or a genetic algorithm, which could be impractical in cases where services rapidly change their QoS parameters.

In [7] the author presents formal approach that allows a workflow architect to develop and evaluate complex composite Web Services in the terms of reliability. The article analyses workflow patterns considering a specific set of patterns from control-flow perspective. Control-flow defines the execution order of a set of activities that are incorporated in Web Service composition. Furthermore, the author narrows the set of workflow patterns to the set of relevant workflow patterns from the reliability point of view. In addition, the authors use pattern-specific formulas in order to express the overall pattern reliability. Also, in the article the authors use reduction algorithm that recognizes workflow patterns in Web Services composition and calculates reliability for that particular part of composition and collapses it into a single task. The process is iterated until the whole Web services composition is collapsed into a single pattern and its reliability is calculated. In this article, the authors did not present the reduction algorithm and did not answer how the reliability factors of the elementary Web Services are calculated.

In [12] authors provide a formal model for predicting the reliability of Web Services. The distinction is made between three types of services: atomic services, tight composite services and loose composite services. For atomic type of services, the presented model uses collected data from service usage history. This requires

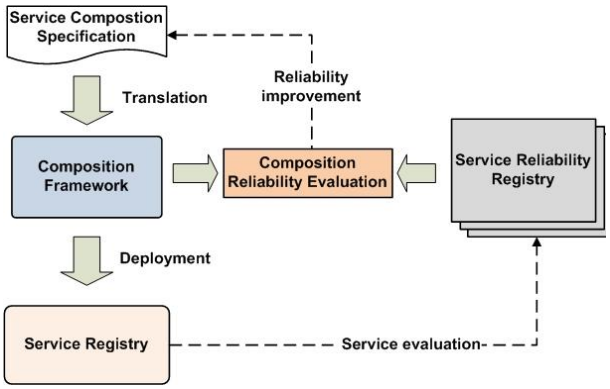

Figure 1. Service Composition Development Model

the extension of the UDDI model, which is presented in the article. The extended model assumes tracking service usage and collecting usage data. For composite type of services, the presented model considers both the overall structure of the composite service and the reliability of each atomic service that is a part of the composition. The model isolates elementary composite structures and defines formulas for their reliability calculation. Authors also describe the recursive algorithm for composite structure reliability computation in pseudo-code. Algorithm assumes recursive substructure recognition and reliability computation until the elementary structures are recognized and their reliability calculated. However, the substructure recognition algorithm is not described in the article.

III. RELIABLE SERVICE COMPOSITION

In this paper we introduce a model for service composition reliability estimation. We argue that this model can be used to aid the service composition development process by improving the overall composition reliability. The presented approach is based on the development model shown in Figure 1. The elements addressed by the model are briefly described below.

Service Composition Specification is a description of composite service's functionality. Specifically, it defines the interaction between various services which is needed in order to achieve the desired functionality for a given composition. Interaction between services is defined as a sequence of service invocations. The services used to construct service compositions will be referred to as atomic services in the rest of the paper. When designing a service composition specification, the necessary condition is to satisfy the functional requirements of the composition logic. However if the functional requirements can be met by constructing more than a *single service composition specification*, the non-functional requirements, such as various quality of service parameters, can be taken into account. In the presented model, service composition specification can be defined by utilizing business process modeling languages, like *BPEL* [11], *CL* [13], *SSCL* [13], *Husky* [14], or other modeling notations. The specification can be generated by humans or automatic service composition frameworks.

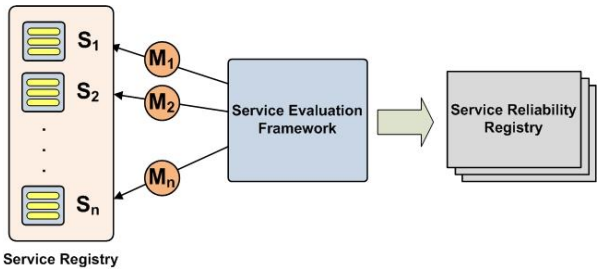

Figure 2. Service Composition Reliability Modeling

Composition Framework translates the *Service Composition Specification* into its implementation. The framework estimates the overall reliability of the service composition based on the reliability of each atomic service. The reliability values of atomic services are accessed from the *Service Reliability Registry*. The constructed reliability model is used to detect the weak points in the composition and suggest improvements to the *Service Composition Specification*. Reliability model generation and composition improvement methods are further discussed in the following sections. Once the reliability requirements are met, the *Composition Framework* is used to deploy the service composition and add it to the *Service Registry*.

Service Registry contains the list of available atomic services and all the necessary information needed to properly invoke their functionality. An example of Service Registry implementation are UDDI based registries for Web Services. It should be noted that compositions can also be registered in the *Service Registry* and used as atomic services in other compositions.

Service Reliability Registry stores the reliability values for all methods of each atomic service. The values are obtained by performing reliability evaluation measurements used to gather information on actual service performance. The reliability measurement for atomic services is discussed in the following subsection.

A. Reliability of Atomic Services

Crucial prerequisite in predicting the reliability of a service composition is the ability to reason about reliability of each atomic service included in the composition logic. In this paper, we consider a service evaluation model presented in Figure 2. Estimation of atomic service reliability is done by performing measurements on the deployed services which are accessible through the service registry. The measurements are performed by the evaluation framework. For each service $S_1 \ldots S_n$ from the service repository a corresponding monitoring process $M_1 \ldots M_n$ is maintained. Each monitoring process is responsible to periodically test the service by invoking all its methods. In the proposed framework, reliability of each method is stored separately. One of the approaches to predict reliability is to maintain the history of the method's invocation attempts. The reliability can then be estimated by dividing the total number of fault occurrences by the number of invocation attempts. The Service Evaluation Framework should provide means to detect Byzantine faults, i.e. both the

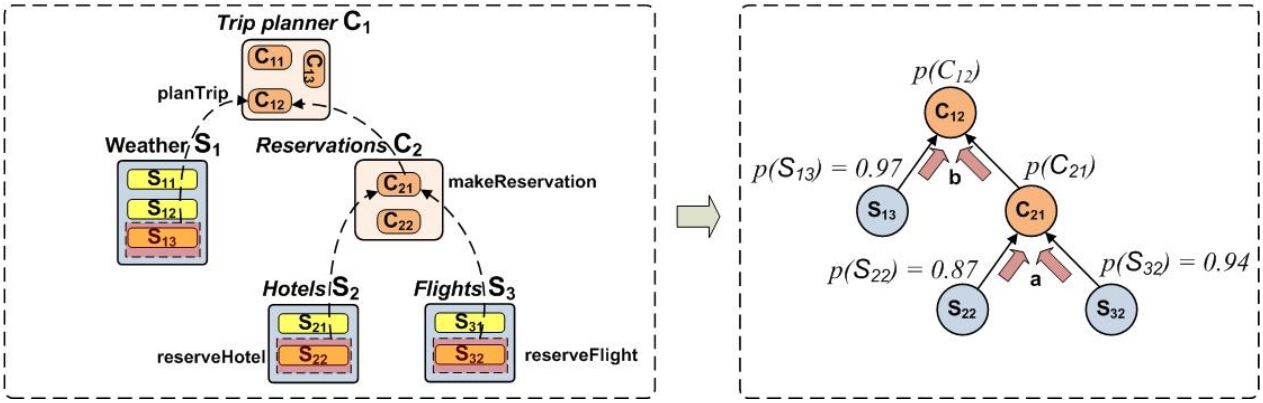

Figure 3. Service Composition Reliability Modeling

faults caused by the lack of service, as well as faults caused by improper service, like returning corrupt data. Therefore, the reliability for a service method can be modeled by the equation:

$$r_{i,j} = (a_{i,j} + c_{i,j}) / n_{i,j} \quad (1)$$

where i is the service identificator, j the service method identificator, $a_{i,j}$ and $c_{i,j}$ are total counts of failures related to availability and service correctness respectively and $n_{i,j}$ the total number of executed tests.

In order to get a precise reliability measurement, the service tests should be relatively frequent, but, on the other hand, they should not have a severe impact on the service's performance itself. In addition, the tests should be run for some time to get a wider usage history. On the other hand, the service responses should be tested thoroughly to detect if the service is operating within its specifications. Therefore, the test coverage should be high and include corner cases. These issues are not in the scope of this paper. For the reasons mentioned above, in would be impractical to strain the service composition designers with the necessity to create and execute service tests before the service composition can be implemented. Therefore the *Service Evaluation Frameworks* should be run by independent third parties whose responsibility would be to maintain the *Service Reliability Registries*. An approach similar to service level agreement (SLA) can be used to construct such an environment.

B. Service Composition Reliability Model

A simple service composition that will be used as a use case example for the reliability prediction model throughout the paper is presented in Figure 3. Specifically, we analyze the reliability of the *planTrip* method exposed by the *Trip planner* composite service C_1. The method is used to plan a trip according to weather conditions at the destination and the possibility of reservations. This is done through *makeReservation* and *getWeather* methods exposed by *Reservations* C_2 and *Weather* S_1 services respectively. It should be noted that the *makReservation* method is exposed by the composite service C_2 that utilizes *reserveHotel* and *reserveFlight* methods of services *Hotels* S_2 and *Flights* S_3 to check if the reservations are possible. Reliability of all the methods exposed by atomic services is previously measured and can be accessed through the *Service Reliability Registry*.

We introduce a model for overall service composition reliability prediction based on the directed acyclic graph approach. When reasoning about service composition reliability, it is clear that the reliability of each atomic service has impact on the overall composition reliability. We further define the model below.

Graph node stands for the reliability of an atomic (composite) service method S_{ij} (C_{ij}), where i is the service identificator and j the service method identificator. The atomic service node has no incoming edges, while the composite service node can have both the incoming and outgoing edges.

Graph edge leading from an atomic (composite) service node S_{ij} (C_{ij}) to composition node C_{ab} denotes that the composition C_a is dependent on service S_i (C_i) since its method b invokes the method j of S_i (C_i). Therefore, the edges are oriented in the causal direction, since they point from the cause (atomic service), to effect (service composition).

It should be noted that cycles in the presented graph model are not allowed. In effect, cycles in service invocations are not possible and are therefore not supported by the proposed model.

An example of the proposed model for the *Trip planner* composite service is given in Figure 3. The method *planTrip* (C_{12}) of the service composition C_1 is modeled as node C_{12} in the directed acyclic graph. Similarly, nodes S_{13} and C_{21} are assigned to methods *getWeather* and *makeReservation*. Since the method C_{12} invokes methods S_{13} and C_{21}, such dependence is denoted by directed graph edges from service nodes S_{13} and C_{21} to C_{12}. Similarly, since the method C_{12} is constructed by composing methods *reserveHotel* (S_{22}) and *reserveFlight* (S_{32}), an edge pointing from nodes S_{22} and S_{32} to C_{12} exists in the presented graph model.

We define a service composition reliability prediction based on the directed acyclic graph. It is assumed that the failure of any atomic (composite) service method S_{ij} (C_{ij})

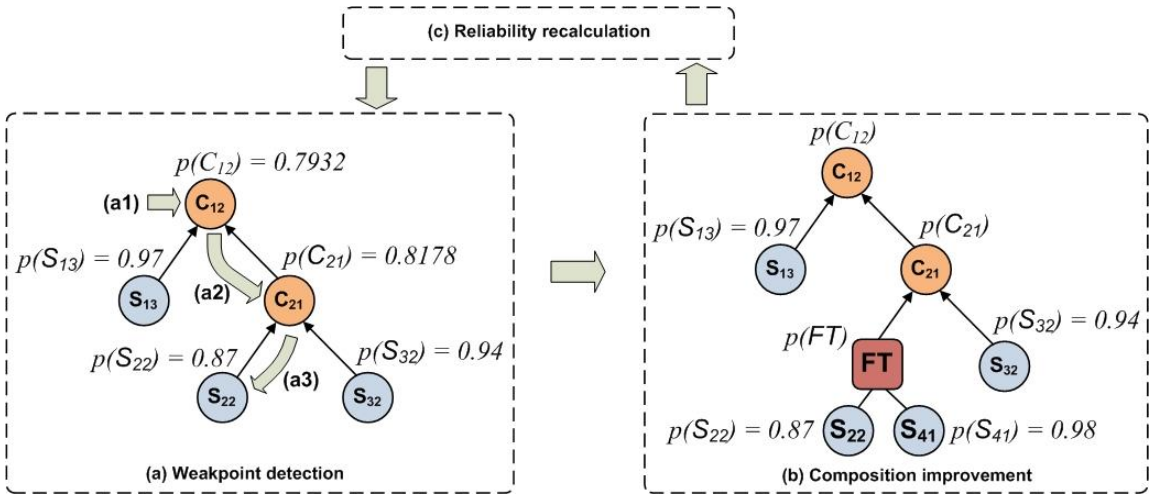

Figure 4. Service Composition Improvement Cycle

invoked by C_{ab} will lead to the failure of the composition method C_{ab}. In addition, we consider all failure occurrences in services S_{ij} (C_{ij}) to be conditionally independent. Therefore, the reliability prediction for a graph node C_{ij} which has n incoming edges $S_{1,1} \ldots S_{n,m}$ can be calculated by the following equation:

$$p(C_{i,j}) = p(S_{1,1}) \cdot \ldots \cdot p(S_{n,m}). \quad (2)$$

Since the atomic service in a service composition can be another composition, the depth of the acyclic directed graph can be greater than 2, as presented in example on Figure 3. Thus, the overall reliability is calculated in the causal direction, starting with the atomic services, i.e. services with no incoming arcs. Consequently, for the example in Figure 3, first the reliability of C_{21} is calculated (a) $p(C_{21}) = p(S_{22})p(S_{32}) = 0.8178$ and then the reliability of the service composition method C_{12} (b) $p(C_{12}) = p(C_{21})p(S_{13}) = 0.7932$.

C. Reliability Improvement Method

The presented directed acyclic graph reliability prediction model can be used to improve the overall service composition reliability. In this paper we introduce a reliability growth model presented in Figure 4. The improvement cycle defined by the model consists of: *weak point detection*, *composition improvement* and *reliability recalculation*. First, a weak point in the model is detected (a). The detected weak point is then improved to increase its reliability (b). Finally the overall reliability prediction of the service composition is recalculated by utilizing the model defined in Section 3b (c). The iterative process of service composition reliability improvement is continued with step (a) until the desired composition reliability has been achieved.

Weak point detection method presented in this paper is based on traversing the directed acyclic graph in the non-causal direction, i.e. opposite of the direction in which the reliability prediction is calculated, starting at the composition node. In each step, the next node is selected by choosing the one of the underlying service nodes with the lowest reliability value. This process is repeated until an atomic service is reached, i.e. a node with no incoming arcs. The located atomic service is considered to be a weak point in the service composition. An example of the presented weak point detection method applied to the *Trip planner* service model is given in Figure 4a.

The entry point for the algorithm is the node of the evaluated service method. In the given example, service composition method *planTrip* is analyzed and thus the node is C_{12} selected (a1). In the next step, node C_{21} is selected over node S_{13} because $p(C_{21}) < p(C_{31})$ (a2). Finally the node S_{22} is reached since $p(S_{22}) < p(S_{32})$ (a3). Node S_{22} is the detected weak point since it has no incoming edges to continue the search.

Once a weak point is detected, the overall composition reliability can be improved by performing modifications to the node in question. A possible solution is to change the corresponding atomic service with a more reliable one, if such a service exists in the service repository. If no such change is possible, we propose the introduction of fault tolerant services. The fault tolerant services are a wrapper around several other services that expose the same functionality. Therefore, fault tolerance is achieved by utilizing space redundancy, meaning that if one of the redundant atomic services fails, others can compensate for this failure. For example, services S_{22} and S_{41} in Figure 4b are redundant.

Several strategies can be used to construct the fault tolerant service *FT* [9][10] which performs both the fault detection and mitigation. For example, if the recovery blocks strategy is used, the reliability of the fault tolerant service can be calculated as follows:

$$p(FT) = 1 - p(\neg S_{1,1}) \cdot \ldots \cdot p(\neg S_{n,m}), \text{ where } p(\neg S_{i,j}) = 1 - p(S_{i,j}) \quad (3)$$

and $S_{1,1}$ … $S_{n,m}$ are redundant atomic services. Once the improvements to the composition are complete, the overall composition reliability is recalculated.

For the service composition example given in Figure 4b, the weak point S_{22} can be strengthened by applying one of the improvement strategies. Let us assume that another service for hotel reservation is available in the service registry and that the reliability of corresponding method is $p(S_{41})=0.98$. We can apply the described fault tolerance approach based on recovery blocks by replacing node S_{22} with a fault tolerant service *FT*. Service *FT* uses both the services S_{22} and S_{41} to improve the reliability by compensating in cases that one of the services fails. The reliability of node *FT* is calculated by utilizing equation (3): $p(FT) = 1- (1 – p(S_{22}))(1 – p(S_{41})) = 0.9974$.

Finally, the overall service composition has to be recalculated. The reliability estimation for node C_{21} equals: $p(C_{21})= p(FT)p(S_{32}) = 0.9375$. Finally, the reliability for method C_{12} is reestimated to: $p'(C_{12})= p(C_{21})p(S_{13}) = 0.872$. Thus, the reliability estimation for the method C_{12} has improved by $p'(C_{12}) - p(C_{12})=0.1443$ as a result of strengthening the weak point S_{22} with a fault tolerant service *FT*.

IV. CONCLUSION

In this paper we presented an overview of methods used to model service composition reliability. A model for service composition development aimed at increasing the composition reliability was introduced. In order to calculate the reliability prediction, it is necessary to know the values for reliability of each atomic service participating in the service composition. A brief overview of an atomic service evaluation model based on the testing framework was given. The composition reliability prediction model is constructed as a directed acyclic graph, where nodes present service method reliabilities and edges composition dependabilities. Furthermore, a method for composition improvement has been introduced. The method applies a simple strategy to locate weak points in a composition. A weak point is detected by traversing the graph and selecting the component with lowest probability in each step until an atomic service is reached. The weak point can then be further improved by replacing it with a more reliable service or by utilizing fault tolerance methods, like *recovery blocks* or *N-version programming*. The iterative process of calculating overall composition reliability and improving weak points is repeated until the desirable reliability level is reached.

The accuracy of the model presented in this paper is influenced by the accuracy of reliability estimation for atomic services. As stated in Section 3a, the accuracy of this estimation depends on the frequency of conducted tests, as well as the test coverage for the faults related to improper service. Furthermore, the assumption that a failure in one service will cause the failure of the entire service composition can cause an overestimation. This is due to the fact that not all the services might be equally crucial for the composition's normal operation. In such cases, our model will estimate the reliability to be lower than the actual value. Introducing weight factors representing the influence of a service on the composition can further improve the model. Thus, the future research efforts will be directed at investigating how efficient and quickly, i.e. in how many steps, the proposed reliability growth model converges towards the desired reliability. Additionally, the application of other improvement methods and their impact on the model will be considered.

ACKNOWLEDGEMENTS

The authors acknowledge the support of the Ministry of Science, Education, and Sports of the Republic of Croatia through the Computing Environments for Ubiquitous Distributed Systems (036-0362980-1921) research project. Furthermore, the authors thank Dejan Skvorc, Miroslav Popovic, Ivan Zuzak, Klemo Vladimir, Ivan Budiselic and Zvonimir Pavlic from the University of Zagreb Faculty of Electrical Engineering and Computing.

REFERENCES

[1] W. T. Zhang, et al., "A Software Reliability Model for Web Services", The 8th IASTED International Conference on Software Engineering and Applications, 2004, pp. 144-149.

[2] Z. Liu, N. Gu and G. Yang , "A Reliability Evaluation Framework on Service Oriented Architecture", 2nd International Conference on Pervasive Computing and Applications (IPCA 2007), October 2007, pp. 466-471.

[3] F. Mahdian, et al., "Modeling Fault Tolerant Services in Service-Oriented Architecture", Third IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE '09), July 2009, pp. 319-320.

[4] Z. Zibin and R. Lyu, "Optimal Fault Tolerance Strategy Selection for Web Services", International Journal of Web Services Research (IJWSR), vol. 8, no. 2, 2010, pp. 41-63.

[5] W. Zhao, "A Lightweight Fault Tolerance Framework for Web Services", IEEE/WIC/ACM International Conference on Web Intelligence , November 2007, pp. 542-548.

[6] P. Sun, "Optimal selection of composite web service based on QoS and risk evaluation", International Conference on E -Business and E -Government (ICEE 2011) , May 2011, pp. 1-4.

[7] L. Coppolino, L. Romano and V. Vianello "Security Engineering of SOA Applications Via Reliability Patterns", Journal of Software Engineering and Applications (JSEA), vol.4, no.1, January 2011, pp.1-8.

[8] D. Booth, et al., "Web Services Architecture", W3C Working Group Note, February 2004, http://www.w3.org/TR/ws-arch/.

[9] B. Randell and J. Xu, "The Evolution of the Recovery Block Concept", Ch. 1 in Software Fault Tolerance, M.R. Lyu, editor, John Wiley & Sons Ltd., 1995, pp. 1-22.

[10] A. Avizienis, "The methodology of N-version programming", Ch. 2 in Software Fault Tolerance, M.R. Lyu, editor, John Wiley & Sons Ltd., 1995, pp. 23-46.

[11] A. Alves, et al., "Web Services Business Process Execution Language Version 2.0", OASIS Standard, April 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[12] B. Li, et al., "Evaluating the Reliability of Web Services Based on BPEL Code Structure Analysis and Run-Time Information Capture", Asia Pacific Software Engineering Conference (APSEC), December 2010, pp. 206-215.

[13] S. Srbljic, D. Skvorc, D. Skrobo, "Programming Language Design for Event-Driven Service Composition", AUTOMATIKA – Journal for Control, Measurement, Electronics, Computing and Communications, vol. 51, no. 4, March 2011, pp. 374-386. December 2010, pp. 206-215.

[14] D. Skrobo, "Service Composition Based on Tabular Programming", PhD thesis (in Croatian), University of Zagreb, October 2008.