2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops

Studying the Propagation of Failures in SOAs

Cristiana Areias, João Carlos Cunha
Instituto Politécnico de Coimbra, ISEC, DEIS
Coimbra, Portugal
cris@isec.pt, jcunha@isec.pt,

Marco Vieira
CISUC, Department of Informatics Engineering
University of Coimbra, Portugal
mvieira@dei.uc.pt

***Abstract*—Although Service Oriented Architectures (SOAs) are being increasingly used in business-critical scenarios, the applicability of Verification and Validation (V&V) is still very limited. The problem is that V&V activities have to be implemented at runtime to fit the characteristics of SOA. Recent proposals of runtime V&V techniques specific to SOA domain are far from being complete and a key issue lies in understanding how the *"failures propagate"* in a dynamic system and how to continuously verify its evolving elements. This paper introduces an approach to deal with the propagation of failures in a SOA environment. The proposed technique is based on three key steps: estimating the failure rate of the individual services, using fault injection to find the exposure of each service to failures from the invoked services, and estimating the impact of each service in the overall architecture. The overall approach is presented with a brief demonstration of its application.**

***Keywords*—failure propagation, fault injection, FMEA, Runtime V&V, SOA**

I. Introduction

Service Oriented Architectures (SOAs) are a paradigm for organizing and utilizing distributed functionalities whose main emphasis is on the loose coupling among interacting services [1]. These services are deployed in a distributed way, and are consumed by other services and applications, frequently over a network. SOAs present particular characteristics as high complexity, extreme dynamicity, and a very large scale of composability. The increasing usage of SOAs in business-critical applications calls for quality assurance approaches that allow continuously asserting their trustworthiness.

Verification and Validation (V&V) is in the foundation of critical applications' development, providing a quality assurance process for checking if a system meets the specifications and fulfills the intended purpose [2]. Besides the functional behavior, it also considers non-functional features such as dependability and security aspects. A plethora of V&V techniques can be used, and although testing is probably the most used, in the context of critical systems other more stringent activities are recommended or even required [2]. One of the most used activities in the context of safety-critical systems is the Failure Mode and Effect Analysis (FMEA) [3].

The typical V&V lifecycle in critical system assumes a structured and highly documented development process that allows gathering the required quality evidences, and presumes that the system does not evolve after deployment. The problem is that this approach does not fit the characteristics of service oriented environments, where a **multitude of services is continuously being deployed, interconnected and updated**, following software development approaches that favor rapid deployment and frequent updates of services. In fact, the dynamic nature of SOAs, together with the demand from organizations for rapid changes in business requirements, result in an overlapping between the design and usage phases.

To overcome these problems, new V&V approaches must be applied at runtime to continuously assure the required quality and thus improve trustworthiness. One of the recent efforts towards this was the proposal of a preliminary approach to apply FMEA in SOA domain [4]. The approach considers the specific characteristics, focusing on the analysis of services and on dealing with the dynamicity of the environment. However, **understanding the effects of service failures** is the first step towards its applicability. Before this, it is not possible to determine the parts of the system that need to be re-V&Ved when some part of the architecture changes, and thus it is not be possible to deal with dynamicity, evolvability and composability of SOAs.

Failure propagation only makes sense in this kind of domain, as there are different system boundaries at different levels. In practice, in a SOA environment, when a service fails, at *1) Service Level* it represents a failure and an interface fault to the invoking service, while at the *2) SOA System Level* it represents an error being propagated.

The idealized approach is based on the following phases: **i) Design phase –** for each individual service, use traditional V&V techniques to estimate its failure rates and also use fault injection techniques to determine its exposure to its invoking services; **ii) Analysis Phase –** considering the architecture, estimate for each service the total probability of each service to fail and also the impact of failures in the remaining services. This analysis can even affect the decisions before deployment of parts of the system; **iii) Runtime phase –** monitor the infrastructure to detect changes in services or in the architecture that require repetition of the process and also to verify if real failure rates match the estimated ones.

Upon completion, this process will allow understanding: *1)* which are the most critical services for the architecture; *2)* which are the services that need more improvement; *3)* which are the services whose improvement would benefit the most the complete infrastructure; *4)* which services must be re-V&Ved when there are changes in a service or in the architecture;

The outline of the paper is as follows. The next section presents relevant background and related work. Section III presents the overall proposal for the preliminary approach while the Section IV presents the details of each phase, the key challenges and the plans to overcome each one of them. Finally, Section V concludes the paper.

978-0-7695-5533-1/15 $31.00 © 2015 IEEE
DOI 10.1109/DSN-W.2015.18

II. Background and Related Work

SOAs are increasingly being used as an approach to make business critical systems more efficient and agile [5]. SOAs are composed by several services, which are delivered through standardized interfaces and message exchanging protocols that create dynamic and distributed software systems. Also, services can be invoked and distributed across several organizations over the Internet allowing the interoperability, flexibility and dynamicity provided by SOA systems [6].

To assure that software achieves the required levels of dependability and security, business critical organizations spend a portion of the system development budget on V&V activities. In fact, the cost to preventively find and remove defects is usually smaller than it is a system failure repair. Even worse, a system failure can have disastrous consequences on the reputation and sustainability of the organization. V&V is the process for checking if a system meets the specifications and fulfills the intended purpose [2]. While **verification checks the conformance to the specifications**, trying to identify product faults or errors, which give rise to failures, validation is used to get the necessary evidences to **ensure that the system satisfies the intended needs** [2].

Several V&V techniques have been applied during software development, ranging from static to dynamic techniques [2], such as *walkthroughs, inspections, traceability analysis and testing* [2]. RAMS analysis usually designate processes that that attempt to assess the system according to its properties of *Reliability, Availability, Maintainability and Safety* and include diverse techniques such as Failure Mode and Effects Analysis (FMEA), its extension to Failure Mode, Effects and Criticality Analysis (FMECA), or Fault Tree Analysis (FTA) among others. These activities aim at evaluating the critical parts of the system and directing the efforts of V&V where consequences of failure are the most severe [2]. Software FMEA [3] is an example of a widely used activity in the context of safety-critical systems.

Looking into SOA context, its highly dynamic nature poses serious challenges on applying traditional V&V techniques. In fact, as the system is always evolving, some configurations can only be seen at runtime [1], meaning that V&V should be applied not only during system development, but also after deployment [7]. The current state of the art on V&V for service-oriented environments focuses mainly in techniques for verifying the correctness of single or composed services across formal methods [8] and testing [1], [9]. Although very useful and effective, these techniques are essentially applied to individual services, based on assumptions that may not hold in SOA environments. Furthermore, SOAs are such a complex environment that it is impracticable (due to time and resource constraints) to apply V&V to all the services at runtime.

Software FMEA offers a way to **systematically analyze all the structure of the SOA environment** and determine which parts are the most critical in order to prioritize the employment of further V&V. An example of the application of a FMECA to a service based system is presented in [10], but the runtime requirements of the system were disregarded, which renders the approach almost useless in this environment. Conversely, the proposal in [4] introduces the use of FMEA as a tool to provide an extensive knowledge of the overall environment of the system through the identification of the dependencies and interactions between services, and ranking them according to their relative risks. After this analysis, more V&V efforts can be directed towards the services with higher risk and also to plan effective integration of new services. As discussed in the paper, the approach still presents *many challenges that must be overcome to become usable and effective in SOA domain* [4].

A risk-based approach for selecting and prioritizing test cases for semantic web services is proposed in [11]. The analysis focuses on two factors of risk estimation, failure probability and importance, and from three aspects, ontology data, service and composite service. This means that the success of this approach lies in the information provided by the semantic web services. Conversely, our approach takes advantage of fault injection techniques that we believe may help us to understand better the behavior of each service, and it is not dependent on the information provided.

The authors of [12] review the mathematical foundations of reliability modeling of a SOA as a function of the reliability characteristics of its basic elements. Although it provides interesting insight, the analysis is limited by the model adopted, a *"fail-stop with no repair model"*. Applying a different approach, the authors of [13] propose a collaborative reliability prediction approach for SOAs, which employs the past failure data of other similar users to predict the Web service reliability for the current user. Although this does not solve the problems that we are addressing here, it provides interesting techniques to help us in overcoming some of the gaps in our approach.

III. Analysis of Failure Propagation in SOAs

Our methodology was designed to fit the requirements of the targeted domain: the three-phase design allows the *system integrator* to apply effective V&V strategies, to analyze the effectiveness of the system, and to be prepared for the problems or changes that can be faced at runtime. At the same time, the cyclic and iterative process is prepared to **commute between offline and runtime contexts**, without requiring the infrastructure to stop. In fact, as will be presented, the approach considers the use of shadowing techniques to analyze the *"to be deployed"* versions of the services without harming the remaining infrastructure. Meanwhile, it aims at estimating the impact of each service in the overall architecture, both through experimental and theoretical analysis that will allow to define boundaries of the parts that need to be re-V&Ved in case some change occurs.

From the perspective of our approach, each service is considered as having one public operation. A composed service, if there is knowledge and access to its internals, should be decomposed in multiple interconnected services. Additionally, the elements as orchestrator and messages queues are also abstracted as services. It is known that, in SOA environments, the systems can be very large, both in size and complexity, with the possibility of even losing the notion of their boundaries. This way, it is necessary that our analysis focuses only on a predefined **scope** that encompasses the services that the interested party has access to and is concerned in performing V&V. Later, we will discuss how we can deal with the services outside of this scope, but most of the necessary information is available in Service Level Agreementss (SLAs).

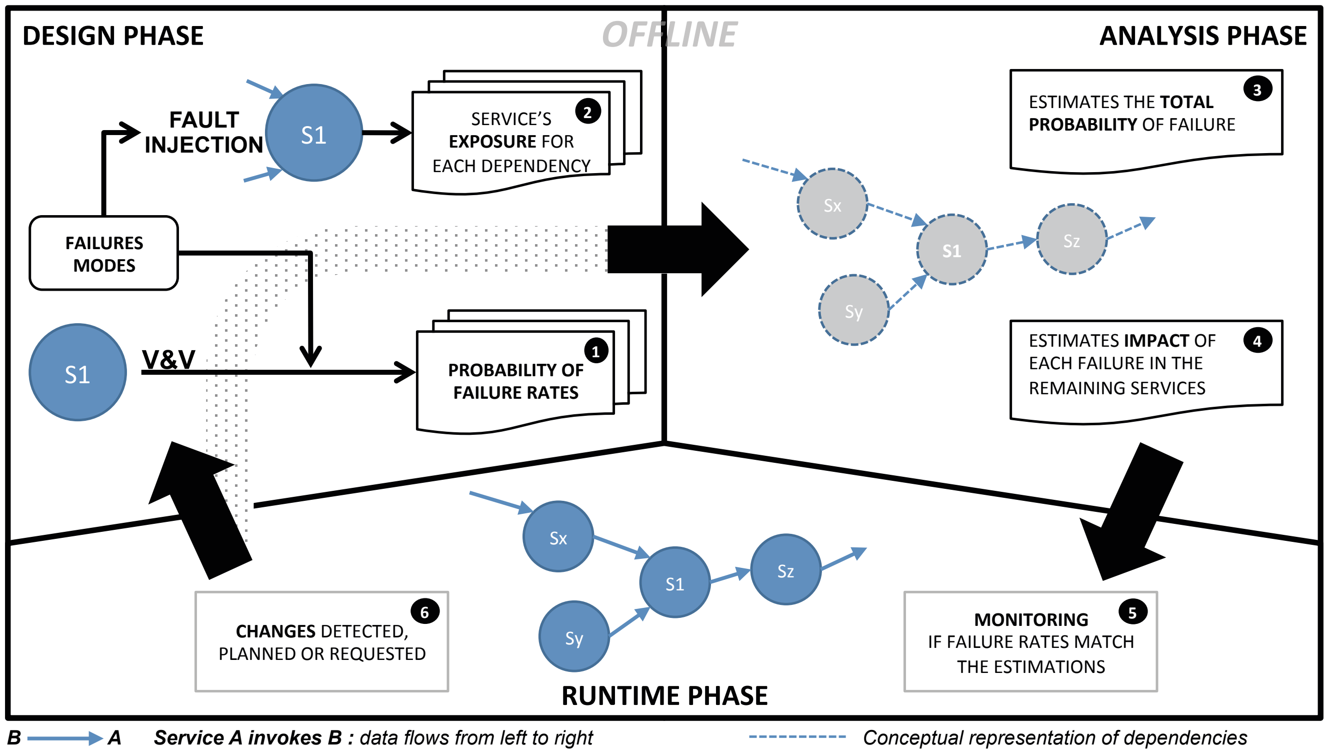

DESIGN PHASE
OFFLINE
ANALYSIS PHASE
FAULT INJECTION
S1
SERVICE'S EXPOSURE FOR EACH DEPENDENCY
2
FAILURES MODES
S1
V&V
PROBABILITY OF FAILURE RATES
1
ESTIMATES THE TOTAL PROBABILITY OF FAILURE
3
Sx
S1
Sz
Sy
ESTIMATES IMPACT OF EACH FAILURE IN THE REMAINING SERVICES
4
CHANGES DETECTED, PLANNED OR REQUESTED
6
Sx
S1
Sz
Sy
MONITORING IF FAILURE RATES MATCH THE ESTIMATIONS
5
RUNTIME PHASE
B A Service A invokes B : data flows from left to right
Conceptual representation of dependencies

Fig. 1. Overall representation of the Failure Propagation Analysis approach and usage lifecycle.

Fig. 1 presents an overview of the preliminary approach for analysis of failure propagation in SOAs. This approach is based in three main phases, each involving several steps, in a total of 6, as depicted in Fig. 1 and described as follows:

Design phase – for each individual service:

(1) Use traditional V&V techniques (tests, RAMS analysis, inspections, etc.) to estimate its individual failure rate;

(2) Use fault injection technique to determine its exposure to each of the services that it invokes;

Analysis Phase – considering the complete architecture, for each service:

(3) Calculate the total probability of failure (considering all services, dependencies, failure rates, exposures);

(4) Calculate the impact of each failure mode in the remaining services;

Runtime phase – and monitor the infrastructure in order to:

(5) Understand if the real failure rates match the estimated ones and when the sample is large enough, use the data to adjust the failure probabilities;

(6) Detect potential changes that require repetition of the process in one or more services;

In case changes are planned, detected, or even hypothesized it is necessary that the new services or the ones with modifications in the implementation go back to the step (1). When the changes are only in the relationships of the architecture and not in implementations, it is necessary only to go back to the step **(3)**.

IV. Strategy to Deal with Failure Propagation

The following subsections present the planned strategy for each of the formerly listed steps, as well as the main challenges that are foreseeable from where we stand now.

Before starting with the first step, it is necessary to define the **possible failure modes** that will be considered for the services, i.e. the way in which they deviate from their correct functioning [14]. Existing lists of failure modes for this type of environments (e.g. [9]) may be used as a basis for the definition of a more specific one.

A. Design Phase

As already mentioned, this phase considers each service individually and independently from the others. However, it is assumed that we have information about its interfaces, including the ones used to invoke the services it depends upon. With this information and some specification, the strategy is to use mocking or stubbing [9] techniques to potentiate the following two steps.

(1) Estimating the failure rates

The goal of this phase is to obtain a table with the probability of occurrence of each of the previously specified failure modes, assuming that every dependency works correctly. There are two alternatives to obtain this probability, as follows.

The easiest one is to assume that these **values are provided** to us. In some cases, this can be the only alternative, as it is in the case of third party services to which we do not have access to the internals, and thus we cannot test to estimate the probability of failure. However, it is reasonable that we have at least SLAs from which we can get important information.

The alternative, more challenging, is to **determine this probability using V&V techniques**. Indeed, using techniques as tests, inspections, code analysis, etc. it is possible to create models that provide us with reliability predictions of the software [13]. However, this option raises some important challenges for which we still do not have solutions:

Challenge #1: *Define which are the necessary activities to perform in order to obtain these values.*

What V&V activities are necessary? Is testing enough, or do we need other techniques? How confident can we be in the obtained results?

Challenge #2: *For testing activities, it is necessary to develop automated testing environments. It is also necessary to determine when to finish testing.*

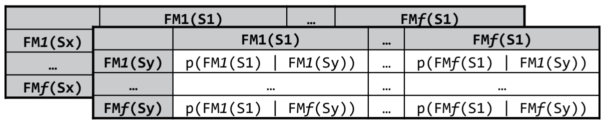
The difficulty here is that it is not possible to provide a generic workload that fits the purposes of every service. Thus, we need dynamic ways to generate different types of workloads, and we must take advantage of the available specification and test cases. Additionally, we can use past failure data to help us configuring the testing activities.

Independently from how the values are determined, during runtime these values can be adjusted based on the real failure rates presented by the services, but only after gathering a significant number of service requests.

(2) Estimating the exposure to other failures

In this step, we will need to estimate the exposure of the service to the failures of the services that it invokes or that it depends upon. By *exposure of S1 to Sx* or `exp(S1,Sx)` we mean the probability of S1 presents a failure mode given that Sx presents a failure mode. As it is possible that a failure mode in Sx causes a different failure mode in S1, the `exp(S1, Sx)` represents a matrix of values with `f*f` dimensions, where `f` is the number of failure modes considered, as explained previously.

In the end, as each service under study has `d` dependencies (in case of S1 in Fig. 1, `d=2`), the output of this phase is a three-dimensional matrix with the dimensions `f*f*d`, as shown in Fig. 2.



	FM1(S1)	…	FM*f*(S1)
FM*1*(Sx)			
…			
FM*f*(Sx)			

	FM1(S1)	…	FM*f*(S1)
FM*1*(Sy)	p(FM*1*(S1) | FM*1*(Sy))	…	p(FM*f*(S1) | FM*1*(Sy))
…	…	…	…
FM*f*(Sy)	p(FM*1*(S1) | FM*f*(Sy))	…	p(FM*f*(S1) | FM*f*(Sy))

Fig. 2. Exemplification of the `f*f*d` matrix obtained in **(2)**.

The challenge here is to obtain the values to fill this matrix. The information about the interfaces between services can easily be discovered from the specification or even from the analysis of the artifacts. With this information, our goal is to create an environment where the service can be inserted and then we emulate its dependencies.

Challenge #3: *Build an automated testing environment that can emulate the dependencies of a service and also inject faults.*

The difficulty here is to create a generic emulating environment that is dynamic enough to emulate dependencies according to the specifications of each service. The strategy will be to create a sandboxing-like system which is able to use mocking and stubbing strategies to emulate the correct dependencies and that can be trained to behave as expected. The final problem is the definition of the workloads and faultloads to submit in each test.

Afterwards, we will be able to easily inject the previously defined failure modes and observe how S1 performs. Basically, we will repeat each failure mode the necessary number of times to have statistically relevant results regarding the behavior of S1, both correct and incorrect. As already discussed, the injected faults intend to emulate failures in the interactions of S1 with other services or resources.

B. Analysis Phase

In this phase a model of the whole system architecture will be considered to calculate the probability and impact of failures at the services' level. Based on the data collected from the previous phase and on the interconnections between the services, we intend to build a mathematical model that allows the estimation of the probabilities of failure of each service, as well as its impact in the other services. This will allow the system integrator to understand which services of the architecture have the highest probability to fail and also which ones have the highest impact in the success of overall infrastructure.

Although the obtained values are just an estimation, they will allow the integrator to take more informed decisions and also to understand how to direct his V&V activities. It will also allow planning of changes in the architecture, as it will allow dealing with hypothesized changes. Basically, the integrator will have at his disposal information that will help him to decide which parts of the architecture should he improve, how much effort should he put on it, and what return of investment could he expect.

(3) Calculate total probability of failure

This step should provide a table containing the probabilities of each service presenting each failure mode, resulting in a matrix of dimension `s*f`, (where `s` is the total number of services, and `f` the number of considered failure modes). These values are computed from the values obtained in **(1)** and **(2)**.

Challenge #4: *Detail the mathematical models to estimate these values based on the ones provided by (1) and (2).*

Although we are still working on this definition, we believe that we can follow a strategy based on Bayesian Networks, similarly to what is presented in [11] to estimate risk.

Challenge #5: *Find ways to mathematically represent more complex constructs as recovery blocks, N-version programming, etc.*

These constructs are hard to represent mathematically, but are unavoidable as they are essential in fault tolerant software. It is thus necessary to study which abstractions better represent them, and to understand if it is possible *to develop models that can be parameterized based on the experimental values obtained in the previous steps.*

The resulting values can help ranking the services by failure probability, allowing the system integrators to understand if the estimated failure rates are inside an acceptable range and, if not, focus on improving them, with special attention those that are more likely to fail. As a corollary, it allows the estimation of the probabilities of failure in the boundaries of the system, meaning that it will be possible to assess the probabilities of failure of the overall system.

(4) Calculate the impact of failures

This step builds on top of the previous to do a much more comprehensive and advantageous analysis. Basically, for each *service under study (Su)* we repeat the previous computation `f` times, setting the probability of a *failure mode under study (FMu)* to `1`. To measure the impact of this change, we compare the obtained probabilities of failure with the original ones, by calculating the difference between the obtained tables and those computed in **(3)**. This difference represents the *variation* in the total probabilities of failure, and thus it is named *delta* (Δ). It is obtained for each service a set of `f` tables similar to those obtained in **(3)**, as shown in Fig. 3.

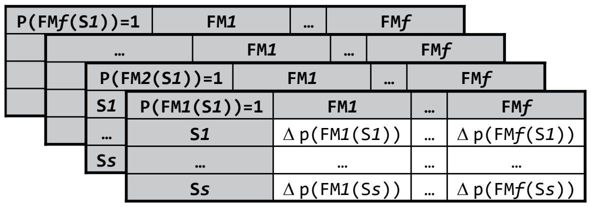


Fig. 3. Delta (Δ) between the probabilities given that a service presents a failure mode and the original ones.

The probabilities obtained represent the impact of each failure mode from each service in the overall infrastructure. As the resulting matrices will contain numbers that will be hard to analyze manually, we are working on metrics that will allow us to reduce the amount of values that we need to analyze.

Again, as we are doing this for each service, by the end we obtain `s` "cubes" like the one represented in Fig.3, with our values organized according to four dimensions: service under study (`Su`), failure mode under study (`FMu`), affected services (`Sa`), affected failure modes (`FMa`). With this data we can proceed to compute metrics using the Δ values and according the multiple dimensions, each with different meaning.

Challenge #6: *Find metrics to characterize the impact of changes in the architecture.*

We are still working on defining the metrics, but these can include at least statistical analysis of the Δ values, including means, standard deviation, etc. We also plan to develop variations of the metrics based on the experience of the final user, i.e. that evaluate the impact of failures in the boundaries of the system.

Analyzing the data by `FMu` allows to predict the most damaging failure modes. In a similar way, analyzing the data by `FMa`, we can predict the most frequent failure modes, guiding us to select which ones the system must be more prepared to tolerate. It is also possible to understand which services are the most sensitive to the failure modes in the architecture through the analysis of the delta values by affected service (`Sa`).

Finally, it is necessary to perform the analysis by `Su`. This may be the most important analysis, as it makes possible to select the services with the highest impact in the overall architecture, understanding how failures propagate and which parts should be re-V&Ved when the `Su` suffers modifications.

In terms of planning and decision support, all this information allows the system integrator to select the most important services for system improvement. Moreover, by estimating the hypothetical probability of failure of a new version of some part of the architecture, the system integrator can calculate its impact in the infrastructure, allowing him to better understand if the planned modifications are the most effective in the concrete scenario.

C. Runtime Phase

This third phase occurs after system deployment, where traditional application of V&V does not apply. This phase copes with the dynamicity of the environment, providing the detection of changes and the consequent context switch for V&V every time some modification is necessary or requested.

In such dynamic and large systems, scalability may become an issue influencing the monitoring process. An efficient monitoring system should be developed as a distributed network of probes that spread in parallel with the services of the SOA providing all the necessary information regarding system modifications.

(5) Monitor real failure rates

This step represents primarily a monitoring activity in which an efficient solution provides relevant information to the integrator. Besides the information regarding system modifications, it should be also collected data about failures and their frequency, which can later be used to check if the real failure rates match the estimated ones and, if not, adjust estimates for future use.

Challenge #7: Gather the relevant data without harming the performance of the infrastructure.

Obviously, it is undesirable to have performance penalties caused by the monitoring activities. Our goal is to develop a parallel infrastructure that is based on distributed set of sensors or probes that feed the data about the running system to a monitor, which processes the activity of the infrastructure. The information collected is then rendered, filtered, analyzed and then stored for future activities. Some techniques to acquire information are already known and consist on using network sniffing or proxy-based solutions.

Challenge #8: Collect large enough samples to allow the adjustment of the failure rate estimates.

Depending on the type of the system and of the considered service, in some situations the generated load may not be frequent of stressful enough to allow gathering enough information to adjust the failure rates. Therefore, it will *be necessary to define thresholds that will allow us to weight the adjustments to be done over time.*

For the services in which significant samples of data are collected, the goal is to adjust the calculated failure rates with the real data This way, there will be a slow adjustment of the failure rates towards the numbers found in the wild. We are still discovering how to do this. We need also to deal with situation when the services with adjusted failure rates are upgraded and thus have new data coming from the **repeated phase of design**.

(6) Deal with changes

Changes in the running system can happen from different ways: *1) planned*, meaning that there is a new version of the software, and the analysis is performed before deploying it into the infrastructure; *2) detected*, meaning that some part internal or external to the system has been modified, so it is necessary to react to these changes; *3) hypothesized*, meaning that there is the possibility that something changed, so this situation must be analyzed before investing resources in dealing with it.

The first scenario is the type that one would expect is more common and that our approach fits better. However, the approach can be very useful in the other two scenarios and should be prepared to deal with them. In every scenario, it is necessary to understand the changes in order to decide which parts of the processes would be necessary to repeat for each of the affected services

Challenge #9: Build effective interfaces to deal with the system and the user, to react accordingly.

The information provided by the monitoring system (5) must be handled properly to automatize some of the processes. However, it is also important to allow the system integrator to visualize and input information in order to take maximum advantage of the system.

Challenge #10: Perform the necessary analysis in the shortest time window possible.

The execution of the process must be as fast as possible, as the system may be waiting to be upgraded or even running without the proper trustworthiness assurance. The problem is that the testing or other V&V activities described in steps (1) and (2) may need some time to be executed.

In operational terms, the hypothesized changes will allow the integrator to take advantage of this approach for planning and evaluation of the cost and benefit of different scenarios.

V. Conclusion

Understanding how failures propagate in SOA environments is essential to adapt V&V techniques to the domain. It allows determining which services must be re-V&Ved when there are changes in a service or in the architecture and also which services need extra attention.

This paper proposes a preliminary approach to do this, using traditional V&V techniques such as testing and fault injection together with analysis and monitoring. Such approach will in the future enable implementation of V&V techniques specific for the domain, and allow that the stakeholders to have more confidence in the SOAs they are using for their business.

Our future steps will focus on overcoming the listed challenges, in such way that we are able to experimentally evaluate the solution as we develop them.

Acknowledgment

This work has been partially supported by the project DEVASSES - *DEsign, Verification and VAlidation of large-scale* (PIRSES-GA-2013-612569, www.devasses.eu), by the project CECRIS - *CErtification of CRItical Systems* (FP7-PEOPLE-2012-IAPP 324334, www.cecris-project.eu), both within the context of the European Union's Seventh Framework Programme (FP7), and the project *ICIS - Intelligent Computing in the Internet of Services* (CENTRO-07-ST24-FEDER-002003, icis.uc.pt), co-financed by QREN, in the scope of the Mais Centro Program and EU's FEDER.

References

[1] G. Canfora and M. D. Penta, "Service-Oriented Architectures Testing: A Survey," in *Software Engineering*, A. D. Lucia and F. Ferrucci, Eds. Springer Berlin Heidelberg, 2009, no. 5413, pp. 78–105.

[2] D. R. Wallace, L. M. Ippolito, and B. B. Cuthill, *Reference Information for the Software Verification and Validation Process*. DIANE Publishing, 1996.

[3] D. Reifer, "Software Failure Modes and Effects Analysis," *IEEE Transactions on Reliability*, vol. R-28, no. 3, pp. 247–249, Aug. 1979.

[4] C. Areias, N. Antunes, and J. C. Cunha, "On Applying FMEA to SOAs: A Proposal and Open Challenges," in *Software Engineering for Resilient Systems*, I. Majzik and M. Vieira, Eds. Springer International Publishing, 2014, no. 8785, pp. 86–100.

[5] H. Tesselaar, "The future of financial services may be banking on SOA," Oct. 2012. [Online]. Available: http://searchsoa.techtarget.com/opinion/The-future-of-financial-services-may-be-banking-on-SOA

[6] G. A. Lewis, "Is SOA Being Pushed Beyond Its Limits?" *Advances in Computer Science: an International Journal*, vol. 2, no. 1, pp. 17–23, 2013.

[7] C. Areias, "A Framework for Runtime V&V in Business-Critical Service Oriented Architectures," in *43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W 2013)*, Budapest, Hungary, 2013, pp. 1–4.

[8] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009.

[9] A. Bertolino, G. Angelis, L. Frantzen, and A. Polini, "The PLASTIC Framework and Tools for Testing Service-Oriented Applications," in *Software Engineering*, A. Lucia and F. Ferrucci, Eds. Springer Berlin Heidelberg, 2009, no. 5413, pp. 106–139.

[10] A. Zalewski, "A FMECA framework for Service Oriented Systems based on Web Services," in *2nd International Conference on Dependability of Computer Systems, 2007. DepCoS-RELCOMEX '07*. IEEE, 2007, pp. 286–293.

[11] X. Bai and R. Kenett, "Risk-Based Adaptive Group Testing of Semantic Web Services," in *33rd IEEE International Computer Software and Applications Conference (COMPSAC'09)*, vol. 2, 2009, pp. 485–490.

[12] V. Cortellessa and V. Grassi, "Reliability Modeling and Analysis of Service-Oriented Architectures," in *Test and Analysis of Web Services*, L. Baresi and E. D. Nitto, Eds. Springer, 2007, pp. 339–362.

[13] Z. Zheng and M. R. Lyu, "Collaborative Reliability Prediction of Service-oriented Systems," in *32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 35–44.

[14] A. Aviienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.