Testing for Highly Distributed Service-oriented Systems using Virtual Environments

F. Nizamic[1], R. Groenboom[2], A. Lazovik[1]
[1] Johann Bernoulli Institute for Mathematics and Computer Science, Faculty of Mathematics and Natural Sciences, University of Groningen
[2] Parasoft Netherlands B.V.

Modern application landscapes are becoming more and more interconnected. The introduction of SOA and ESB technology has increased the dependency on other systems, both internal and external. This has severe consequences for testing SOA applications, their availability, and the constraints on different environments.

1. Testing challenges of service-oriented systems

On March 11, 2009, twenty five residents of a little municipality in the West of the Netherlands unexpectedly received a certificate that they were married or had registered a newborn. What happened? To test the central information system of the country, data in the central people registry has been modified. While the data was supposed to be only a test for the system, it ended up updating the real people's database and initiating the processes for informing the involved citizens. In fact, the change propagated even further: from the central people registry it also affected the information system of the tax office and of the retirement administration.

Currently, testing is facing numerous challenges. From the perspective of a tester, the most of them come from how to prepare and execute tests in shortest amount of time and without a dependency on other processes. In order to have this ability, dependencies such as dependency on testing environment availability or dependency caused by sharing resources with others must be removed. Moreover, testers must start preparation of test scripts before a code that they will test is written. One of the reasons for this is that many software companies are working per Agile working methodology, which means that iterations in which developers and testers need to deliver an additional value to a software are much shorter. If a testers could start the preparation of a test scripts from day one of the sprint, their chances to prepare a tests on time would significantly increase. In addition to that, a testers usually have difficulties to produce all desired states of the external services that are part of their chain testing. In order to cover all code paths and to ensure that system is fully functional tester need to have ability to mock some of the services or, in other words, simulate desired behavior of external services. Furthermore, external services can be available just for a limited time or not available at all.

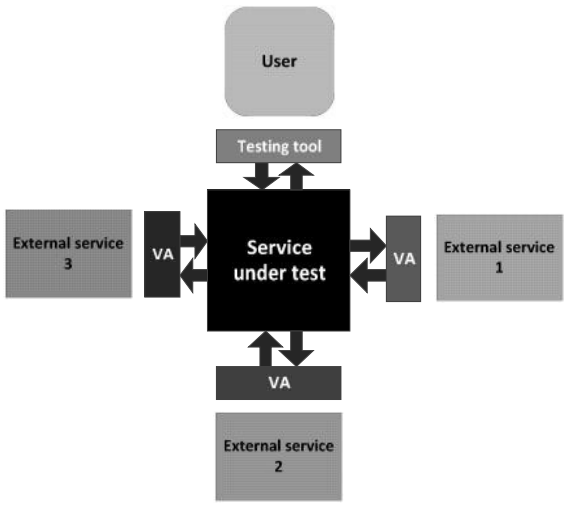
From a perspective of a company, besides quality of a software, mayor concerns are additional costs. That costs can be caused by acquiring additional equipment or human resources. Even higher additional costs lay in maintenance work that was unnecessary or in project delay which occurred; and that is what needs to be reduced or if possible totally eliminated.

In short, all the mentioned issues are pointing at the same goal, and that is: to increase the speed of a testing process and that way reduce the costs by removing dependencies and gaining the control over a testing environments. Most of above mentioned issues could be avoided if a simulation environment, behaving in the very same way as a real system, is used. Nowadays, generation of this kind of simulation environments is very feasible and it is something that will be further discussed.

2. The solution is in a virtualization

The facts are that the software is continuously evolving and that in service-oriented systems the functionality is spread across the network. However, testing of the software still implicitly assumes full control over a system. Reality is that *service under test* depends on the services provided by *external services* (which are not owned by subject performing testing) in order to provide its own service.

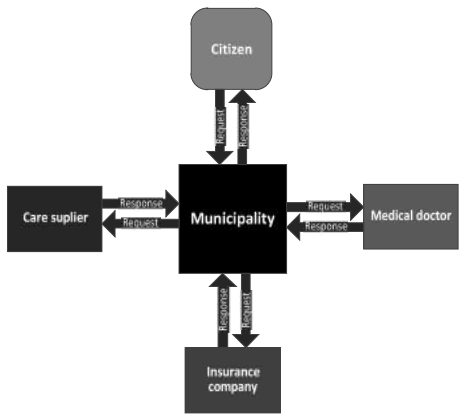
In this work we propose that simulation environments should be semi-automatically generated and that way gained control over an external services and their data. Why?



As a part of integration testing, different versions of interdependent services need to be tested together as the one system. That is because it needs to be confirmed that all functional aspects for this particular combination of services are covered. To prove that, data that is stored in each system need to be appropriate, so all important test cases can be executed. If external systems are not owned by us, which is mostly the case, tests can be performed only with data that is available. Alternatively, virtual assets that mimic a behavior of external services can be created. A set of this kind of virtual assets, representing all external systems, would form a *simulation environment*. It is possible to generate this kind of environment by using currently available tools. Stubs or mock services are not something new and nowadays they can be easily generated just by having a WSDL, log file or by some other way. Once virtual assets are generated/deployed, behavior of external service methods needs to be modeled. To make it even easier, only the behavior of certain methods which are needed in order to cover defined test cases needs to be modeled. This is where the precious time is being saved. A behavior of a services can be modeled simply by defining expected outputs for certain inputs. However, this is not that simple for stateful services. Example of this kind of service methods are the ones that do not have the input parameters and need to return an output (for example *createAccount* method). For this kind of methods, it is difficult to map a set of inputs to set a set of outputs.

Besides generation of simulation environment, whole ecosystem should be populated with corresponding test-data. That should be done only one time. After we have prepared environments that are populated with data that describe their behavior, real tests can be executed on *system under test*. During execution, simulated external services will return a responses that are exactly the same as the one that would be returned by real external services. This way we solve the problem of availability, dependency and maintenance as the simulated systems, which can be easily maintained without a need for intervention of environment administrators, are created.

3. WMO case study



In order to demonstrate how this could work in practice, we present a simulation environment of real business process for the implementation of *Wet Maatschappelijke Ondersteuning Law* (WMO). The WMO is a Dutch law for supporting people that have a chronic disease or disability, so that these

people can independently live in their homes and actively take part in everyday life despite their physical limitations. The support that is provided by the WMO typically includes transportation, a wheelchair, or a home modification (e.g. removing door posts for people using wheelchairs). The responsibility for a WMO request lies with the Dutch municipalities; they handle the complete business process. The process accesses external parties, e.g., insurance companies and doctors for medical advice. In WMO ecosystem, all of the before mentioned issues can be noticed. External services such as *"Medical doctor"* are not available all the time and especially not available for sending test-data which would pollute real database or send a notifications to the real doctors. Besides that, in case of testing *"Municipality"* service, system engineer would need to deploy the latest version of the *external services code* on a test environments. Firstly, question is if we have access to that code that is external to our organization. Secondly, we would depend from some other team (i.e. Ops team) for deployment. And thirdly, question is do we have available testing environment at all. All of the issues could be avoided if we would simulate all external services and model their desired behavior.
For this paper, we have used *Parasoft Virtualize* tool to create virtual assets which would represent all three external services (*Medical doctor*, *Insurance company*, *Care supplier*). *"Citizen"* was simulated by *Parasoft SOAtest* which we used to send the requests to *"Municipality"* service. With this, we shown that it is possible to create all needed components by using currently available tools. We have also modeled virtual assets' behavior to cover basic scenarios and recorded desired requests and responses. Next step would be to find a way how to semi-automatically generate all those virtual assets, populate them with the data and keep up-to-date.
As this was in a way a toy example, now we are approaching companies for appropriate systems that could be used as the real case study. We are aware that we have just scratched the top of an iceberg. Big idea would be in the end to have Agile support for TestOps team which would enable simple switching between staged environments by merging testing and infrastructure knowledge.

About the authors

Faris Nizamic is PhD student at the University of Groningen (RuG). His research interests cover the areas of software testing; service-oriented, distributed and cloud computing. He holds a MSc in Computing Science from the University of Sarajevo, Bosnia and Herzegovina. Last two years he has worked as a Senior Quality Assurance Engineer in NAVTEQ with the main focus on testing of web services and automation of testing processes.
Email: f.nizamic@rug.nl, Web: http://www.cs.rug.nl/~faris

Rix Groenboom is consultant for Parasoft in the area of testing and virtualization of modern SOA, SaaS and cloud-based architectures. He has written a large number of technical articles and presented on many IT industry conferences in Europe and the USA. His core area of expertise is specification, design and validation of software applications. He holds a MSc and PhD in Computing Science from the University of Groningen and published a thesis focusing on the formalization of domain knowledge.
Email: rix.groenboom@parasoft.nl

Dr. Alexander Lazovik is an assistant professor at Distributed Systems group at University of Groningen, after being ERCIM fellow at CWI (NL) and INRIA (F), intern at IBM TJ Watson. He obtained the PhD in computer science from the University of Trento (I) in 2006. His research interests are in the areas of service-oriented and distributed computing. Since last few years, he is also actively interested in pervasive and ubiquitous computing, with an emphasis on dynamic coordination of context-aware embedded devices.
Email: a.lazovik@rug.nl, Web: http://www.cs.rug.nl/~lazovik