Runtime Integration and Testing for Highly Dynamic Service Oriented ICT Solutions – An Industry Challenges Report

Michaela Greiler Hans-Gerhard Gross
Software Engineering Research Group
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
{m.s.greiler|h.g.gross}@tudelft.nl

Khalid Adam Nasr
Logica
Prof. Keesomlaan 14
1180 AD Amstelveen, The Netherlands
adam.nasr@logica.com

Abstract

Modern Information and Communications Technology (ICT) solutions are often widely distributed and highly dynamic service oriented architectures (SOA) with stringent availability requirements. Availability implies that SOA must be reconfigured, updated and maintained during runtime, while retaining their overall operational integrity. This requires that much of the adaptation, integration, configuration and testing activities typically performed offline, during development time, now have to be done online, during runtime. Current component-based runtime platforms such as SOA realize the technological foundations for runtime reconfiguration and maintenance. However, because software engineering methodology has not kept pace with the rapid leap forward in platform technology, adequate methods, techniques and tools for dealing with runtime integration and testing are not yet available. This paper discusses the industry challenges and open issues of integrating and testing SOA infrastructures during runtime.

1 Introduction

Modern ICT solutions are often widely distributed and highly dynamic systems-of-systems (SoS). They provide critical backbone infrastructure for organizations, and are, therefore, increasingly subjected to high availability and dependability requirements. High availability implies that SoS must be reconfigured, updated, and otherwise maintained during runtime, while retaining their overall operational integrity. Additional requirements comprise fast adaptation to changing business processes and new context conditions, such as changing standards, new legislation, or company mergers. The dynamic nature and continuous operation of modern ICT infrastructures demands that much of the component integration, configuration and testing activities typically performed offline, during development time, now have to be done online, during runtime.

Many organizations are in the process of migrating their systems-of-systems to SOA, in order to remain competitive in a rapidly changing world, meet the demands of flexibility and business efficiency, control complexity of current IT infrastructure, and minimize development and maintenance costs.

An IDC study on companies having successfully implemented SOA in their businesses, indicates a range of benefits from SOA [15]. SOA provides higher flexibility by enabling systems to cope with changing business requirements more quickly and with less effort, by reusing components or services that already exist. Integration of new components in the existing system is facilitated. SOA increases business agility, by simplifying the adaptation process, and enabling enterprises to respond quickly to dynamic business challenges. New services can be rolled out easier, and solutions can be brought to the market at shorter lead time. Reusability of code and services, but also business logic and data models, decreases development costs and reduces risks through reconfiguration. Because of the loosely coupled nature of SOA, it becomes easier to integrate solutions from different business partners, facilitating cooperation.

Current service oriented architectures realize the technological foundations for runtime integration and configuration, act as enabling technology, and account for many benefits. However, software engineering has not kept pace with the rapid development of SOA technologies. In particular, adequate methods, techniques and tools for dealing with runtime integration, and the associated quality assurance tasks, pose many problems in practice.

This paper compiles challenges and issues encountered in evolving and testing SOA during runtime. The issues presented partly come from the literature on SOA, and to a large extent from our experience in helping a number of enterprises to migrate their existing ICT infrastructures to SOA, and extending them. Section 2 describes the main challenges that the IT industry is facing when it comes to integration and testing of services in a dynamic SOA environment. Section 3 concentrates on the concrete issues derived from the challenges in terms of research questions, which can be related to managerial and technical problems. Section 4 presents related work, and finally, Section 5 summarizes and concludes this industry challenge article, and gives an outlook on future work.

2 Runtime Integration and Testing for SOA – Industry Challenges

Through loose coupling of services, SOA provides the ideal environment for reuse of existing sub-systems or components, and for enhancement of service capabilities that were not foreseen in the past. To a certain extent, they also provide mechanisms to support reconfiguration during runtime, which is relevant to many systems in industry, e.g., surveillance, transport, enterprise webs, online banking, trading. Platform support is partially available for stopping and resuming services, runtime binding and un-binding, versioning, and service enhancement. That way, large distributed systems can evolve over time, through adding new services and removing obsolete ones, and reconfiguring the bindings between the services during runtime. However, these properties account for a number of emerging challenges.

Stakeholder separation. None of the stakeholders in a SOA is its (complete) owner [9]. This leads to a number of managerial issues, such as lack in communication, restricted influence on system evolution, uncertainty of service usage within the overall SOA, or lack of an overall quality assurance strategy. There are also technical issues. Service developers have access to the code of a particular part of the system, e.g., for performing a full (white-box) unit test, but not to that of the SOA in which their services are used. Here, the services require testing according to the specification of the integrating system [12]. Moreover, the service provider must guarantee fulfillment of service contracts when amended, while the service customer expects seamless availability of the same service. Issues to be addressed come from runtime discovery and binding of services, such as "what is the effect of integrating a new service?", "is an updated service still compatible with the others?", "are the clients of a service notified of any changes?", "what can be (re-)tested if the SOA is reconfigured?", "how does runtime testing affect the production system?", "what is the overall reliability of an evolving SOA?"

Service Integration. New business logic is introduced through new components making up the services. Traditionally, they are developed and tested in a development environment and then deployed in a production environment. However, typical ICT backbones are so complex and expensive that they cannot be replicated easily for testing, and new services, likely to behave differently in a new context, cannot be assessed adequately in a separate testing environment. In most industry-scale SOA, interactions are so complex that the services can only be tested online in the context of the running production environment. This creates a number of interference challenges during integration and testing. One of the techniques required is test-isolation [6, 11], separating testing from nominal operations, and managing runtime resource allocation. Ownership and access limitations imply black-box testing for services when they are integrated during runtime, and require a test-awareness infrastructure [6, 11, 13]. The main challenges to be addressed are "how much of the test-isolation and test-awareness techniques can be provided by the SOA platform to be reused, and how much must be managed by the services themselves?", imposing effort on the service developers.

Service Versioning and Migration. Updating existing business logic and guaranteeing continuous service availability, is another challenge, e.g., in traffic control or multi-tenant systems, which requires support for different versions of the same service, as well as for migrating from an old obsolete service to the new, updated version.

From our own experience, service versioning is not readily done in industry. Old services are replaced by new ones, causing rippling effects throughout the entire system. As a result, clients cannot check the new services without interrupting their own operation. Services should be updated gracefully, giving clients the time to assess and update their own states. Only if the SOA is stable with a new version, should the old one be removed.

Common practice in industry is that changes to the service implementation that are not conspicuous in the interface, are often handled as minor changes, and are not notified to service customers. The service provider justifies this by claiming that only the service implementation has changed and that the updated service should behave as the previous one, which is often not the case. For example, a service providing bank account information is specified through the Web Service Description Language (WSDL), describing two return values; (1) a String with the name of the account owner, and (2) an integer as account number. If the service implementation changes, it could be that the service behavior still complies with this specification, but that it returns the name as "lastname firstname" instead of "firstname lastname". This minor change might not have an impact on the service provider, but it can well make a huge difference for other parties. Further, state migration has to be addressed in a way that service provider and customer always experience a consistent system. Major questions to be tackled in this realm are "what kind of changes, and how should changes be broadcast in a SOA?", "how should regression testing be organized during runtime, and how can it be optimized with respect to the performance of the whole system?"

Service Binding and Reconfiguration. Beside integration of new services, runtime reconfiguration may also take place according to capacity planning strategies, load balancing, Quality of Service or security requirements. Binding occurs at runtime, without notification and without the consumer being able to influence the updating process (ownership issues), and without permitting proper adaptation to the new configuration. The consumers affected by a reconfiguration should be put into the position to react to changes of this kind. This requires a way to assess the likely impact of reconfiguration, and to notify the right other services affected, so they can employ their own built-in counter-measures.

Questions related to these challenges are "how can reconfiguration and late binding be addressed during runtime?", "what are good mechanisms to assess reconfiguration impacts?", "what are good reaction strategies for the services?", and "how can testing facilitate correct binding?"

3 Directions for Addressing the Challenges

Some challenges related to technical issues described above can be addressed through support technologies built into SOA. Mechanisms for installing, starting, updating, stopping and uninstalling services during runtime are available for some frameworks, even without having to restart the platform, such as the IBM WebSphere suite[1], or OSGi [1]. Additional functions exist for service discovery and late binding, and for runtime service orchestration, which is supported through a centralized registry. Version management and explicit definition of service dependencies are also supported by some platforms [1]. However, the methodical and technical issues of SOA runtime testing have not been discussed in the literature.

3.1 Methodical Issues

Even though, many methods and techniques are readily available, their transfer to industry is cumbersome. Such methods comprise built-in testability infrastructure in support of runtime testing and monitoring, and verbose services providing additional query interfaces [14]. Industry is not aware of such techniques, or transfer strategies are lacking. For example, built-in testing infrastructure is not often used in industry, so that stakeholders cannot (re-)assess compatibility and conformance of services with their respective WSDL documents in situ of the running system. But this is also a technical problem, in that platforms do not incorporate even the most basic support for built-in testing, so that organizations have to build such infrastructure themselves. Therefore, an important research direction is represented by questions concerning "which built-in testing infrastructure can be allocated to a platform, and how much must be provided by services themselves?", and "how should this be managed within an overall quality assurance plan?"

Also, stakeholder separation and system ownership has to be resolved through clear protocols of communication and collaboration between different parties. Developers are not directly tied to customers, nor to system integrators, leading to unclear roles and responsibilities in SOA. Research questions are "what are the responsibilities of the service provider, integrator and the customer/user?", "who decides on system evolution, and who implements them?", "how can the customer gain influence in the updating process?", "how to capture evolution requirements?", "which mechanisms are required to propagate system changes?" Industry has to change attitude from merely building systems to conducting and managing them, while increasing software quality and reliability, and providing customer satisfaction. The business and software processes dealing with integration and evolution, have to be reconsidered and adjusted towards better provider-integrator-customer communication, clarifying responsibilities, duties, and exercise of influence.

One might believe that decoupling of services causes decoupling of the stakeholders, e.g., in that the customer does not care about a specific service implementation. However, the opposite is true. For system-level quality assurance it is essential that the stakeholders are tightly coupled, which opens up another issue, i.e., that of trust and privacy or non-disclosure. Here, the primary concern is "how to manage access to system parts for testing and assessment without violating the intellectual property of a service provider, or the privacy of a customer?"

3.2 Technical Issues

Extended Lifecycle Support. Even though SOA platforms already provide lifecycle support of components, we believe this needs to be enhanced. Installation, update, and deletion can be done at runtime, however, update actions may be implemented as a combination of stopping and uninstalling a component, and then re-installing and starting a new one [1]. This renders all registered services unavailable, with effect on all dependent services that cannot fulfill their own obligations. The same holds for service orchestration. Process execution chains may be enhanced during runtime, thereby influencing the process chain, although, currently more elaborate changes to the process execution, like adding a whole new option branch, cannot be done at runtime. Lifecycle and orchestration issues lead to research questions such as "how to organize a non-interfering update process?", and "how to enable complete runtime process updates?" Future requirements towards runtime testing go beyond what is currently feasible.

Central Auditing Authority. In addition to a passive service registry, such as an Universal Description, Discovery and Integration (UDDI) registry, a SOA environment should provide an active auditing authority [5]. Further we argue that the central authority should not only manage service registrations, but also propagation of reconfiguration events, and conduction of runtime testing activities. This is because the service registry is not typically aware of all service user requirements. Research questions in this realm comprise "how should a more intelligent registry be designed?", and "how should system evolution and testing requirements and events be propagated?"

Required Additional Information. The ability and the quality of testing are strongly influenced by available information, and its quality in terms of formalization degree,

[1]`http://www-01.ibm.com/software/websphere/`

completeness and consistency. Dependencies between services have to be described explicitly to facilitate integration and regression testing. Nevertheless, WSDL has major deficiencies in describing dependencies between services and invocation sequences [21]. Such explicit dependencies are particularly important for runtime testability analysis [11]. A primary question to address this challenge is "how can dependencies be made explicit between individual services?"

State Transfer. The transfer of states is still an open issue. From a testing perspective, a service is seen as stateful if the result from a service invocation is not exclusively dependent on the input parameters, so that it delivers a different result depending on its previous usage. The main questions are "how can states be transferred to the new updated version of the component?", "do they have to be transferred, anyway?", and "how can states be synchronized for two versions of the same service?"

Automated Test Generation. By definition, runtime testing must be automated, because it will be initiated automatically through reconfiguration events. In order to being able to check service integration properly, integration and regression test cases should be derived during runtime as well. Services should provide built-in tests and also models describing their provided and expected behavior. Integrating a component requires awareness about dependencies between a new component and the framework in which it will operate, as well as explicit descriptions of service compositions. The operational sequence of service cooperation and the composition of services should be described explicitly in a way that functional accordance can be tested against it [18]. Monitoring the behavior of previous correct service executions can be used for testing purposes for functional equivalence testing. Questions to be addressed are "which artifacts and information are indispensable to enable automated test generation?", or "how can reverse-engineering of models during runtime improve automated testing of SOA?"

Runtime Testability. Since runtime testing is performed while the system is operational, the nominal production-execution must not be affected by testing activities, requiring test-isolation and test-awareness [6, 11] which are also related to test-sensitivity. Testing communication with a service can have effects on the production execution, e.g., by changing the internal state of a component or by producing undesired side effects. Test-sensitive services have to be aware of the fact that they are tested, in order to separate testing from nominal messages, which is referred to as test-isolation. Test-isolation could require simulating parts of functionality or disabling functions during runtime testing. Test-isolation inherently changes the nominal behavior of a system, leading to a number of new issues to be addressed. Questions are "what are adequate test-sensitivity and isolation-techniques for SOA", and "to which extent can they be provided by the SOA runtime platform?"

Performance Overhead. Runtime testing also creates system performance issues. Ideally, the performance of the nominally executing system should not be hampered by runtime testing, and a number of strategies may be used. Regression tests can be minimized based on detailed information about reconfigurations, and by specifying likely rippling effects accurately. The literature on testing commercial-off-the-shelf components (COTS) can provide valuable pointers for testing SOA [8]. Service provider or developer cannot test a service within its deployment context, and they can not foresee all possible deployment scenarios. Aggravating circumstances come through limited access to the service (source code, documentation) because of intellectual property issues. Classical white-box testing approaches are not feasible [4, 6, 13]. Changes to service implementation have to be explicitly made public to facilitate regression testing, and, therefore, reduce performance overhead at runtime. Questions to be tackled in this area are "how can test results be reused to optimize regression testing?", and "how can test orders influence and reduce testing overheads?"

The discussed topics challenge industry especially because existing technologies and methods that would be applicable are not used at all. Knowledge transfer can help to address challenges like the lack in information, automated test generation and runtime testability. On the other hand, challenges like state transfer require further basic research.

4 Related Work

To the best of our knowledge, runtime testing of service centric software has not been studied in depth. Brenner et al. present strategies for testing web services online, based on state behavior [6]. Testability, prerequisite for runtime testing, is studied by Tsai et al. [19], with particular focus on SOA, and by Gonzalez et al. [11], with focus on component-based systems, in general. Still, many open research questions remain in this area, as outlined in this article.

In a broader context, [16, 23] concentrate on runtime testing of component based-software, and provide valuable input also for future research in the area of highly dynamic SOA. A first step towards test reconfiguration based on dynamic changes in SOA environments is presented by [2].

Other related work focuses on testing SOA applications during development and deployment, and highlights web services as implementation option. However, they do not consider testing during operation time. Bucchiarone et al. discuss general problems during integration testing of web services [7]. Integration testing is addressed by Tsai et al. [20], presenting Coyote, an Extensible Markup Language (XML) based integration testing framework; by Bartolini et al. [3], discussing testing of data-flows in web service compositions; and Bertolino et al. [5]. Testing of SOA, in general, is discussed in [8, 9, 10, 22]. Many used concepts are borrowed from the distributed domain [17].

5 Summary, Conclusions, and Future Work

Modern ICT solutions must be reconfigured and updated at runtime, so that much of the component integration, configuration and testing activities now have to be done online as well. In this paper, we presented and discussed the primary industry challenges of runtime integration and testing compiled from our experience with working in the domain of SOA and based on the literature. We conclude that runtime evolution and testing of SOA can only be implemented successfully if, especially, the communication between stakeholders, and the test-isolation and test-awareness of services are improved.

The primary issues that we have planned to address are (1) "which information is provided by industry to enhance runtime testability?", (2) "to which extent can it be used to automate the testing process?" and/or (3) "how can other mechanisms and techniques be applied to cope with the lack of information, e.g., reverse engineering of models from executions?" That way, we intend to advance the state of practice in runtime evolution and testing of highly dynamic service oriented ICT solutions.

Acknowledgments. This work has been funded by the Dutch Government through the Jacquard program (www.jacquard.nl).

References

[1] O. Alliance. *OSGi Service Platform Core Specification*, release 4, version 4.1 edition, 2007.

[2] X. Bai, D. Xu, and G. Dai. Dynamic reconfigurable testing of service-oriented architecture. In *COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference - Vol. 1- (COMPSAC 2007)*, pages 368–378, Washington, DC, USA, 2007. IEEE Computer Society.

[3] C. Bartolini, A. Bertolino, E. Marchetti, and I. Parissis. Data flow-based validation of web services compositions: Perspectives and examples. pages 298–325, 2008.

[4] B. Beizer. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[5] A. Bertolino and A. Polini. The audition framework for testingweb services interoperability. In *EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 134–142, Washington, DC, USA, 2005. IEEE Computer Society.

[6] D. Brenner, C. Atkinson, O. Hummel, and D. Stoll. Strategies for the run-time testing of third party web services. In *SOCA '07: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, pages 114–121, Washington, DC, USA, 2007. IEEE Computer Society.

[7] A. Bucchiarone, H. Melgratti, S. Gnesi, and R. Bruni. Testing service composition. In *Proceedings of the 8th Argentine Symposium on Software Engineering (ASSE'07) Mar del Plata, Argentina*, pages 29–31, August 2007.

[8] G. Canfora and M. Di Penta. Testing services and service-centric systems: Challenges and opportunities. *IT Professional*, 8(2):10–17, 2006.

[9] G. Canfora and M. D. Penta. *Software Engineering*, chapter Service-Oriented Architectures Testing: A Survey, pages 78–105. Springer Berlin / Heidelberg, 2009.

[10] S. Dustdar and S. Haslinger. Testing of service-oriented architectures a practical approach. In *Object-Oriented and Internet-Based Technologies*, 2004.

[11] A. González, É. Piel, and H.-G. Gross. A model for the measurement of the runtime testability of component-based systems. In *5th Workshop on Advances in Model Based Testing (A-MOST 2009)*, pages xx–xx, Denver, Colorado, Apr. 2009. IEEE Computer Society (to appear).

[12] H. Gross. *Component-based Software Testing with UML*. Springer, Heidelberg, 2005.

[13] M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 2008. in press.

[14] M. Momotko and L. Zalewska. Component+ built-in testing: A technology for testing software components. In *Foundations of Computing and Decision Sciences*, 2004.

[15] S. Rogers. A study in critical success factors for SOA. Technical report, Sponsored by Hewlett-Packard, http://www.cio.com/documents/whitepapers/ 6IDCSuccessFramework.pdf, September 2008.

[16] D. Suliman, B. Paech, L. Borner, C. Atkinson, D. Brenner, M. Merdes, and R. Malaka. The morabit approach to runtime component testing. In *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, pages 171–176, Washington, DC, USA, 2006. IEEE Computer Society.

[17] A. S. Tanenbaum and M. V. Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[18] W. T. Tsai, Y. Chen, and R. Paul. Specification-based verification and validation of web services and service-oriented operating systems. In *WORDS '05: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 139–147, Washington, DC, USA, 2005. IEEE Computer Society.

[19] W. T. Tsai, J. Gao, X. Wei, and Y. Chen. Testability of software in service-oriented architecture. In *COMPSAC '06: Proceedings of the 30th Annual International Computer Software and Applications Conference*, pages 163–170, Washington, DC, USA, 2006. IEEE Computer Society.

[20] W. T. Tsai, R. Paul, W. Song, and Z. Cao. Coyote: An xml-based framework for web services testing. In *HASE '02: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering*, page 173, Washington, DC, USA, 2002. IEEE Computer Society.

[21] W. T. Tsai, R. Paul, Y. Wang, C. Fan, and D. Wang. Extending wsdl to facilitate web services testing. In *HASE '02: Proceedings of the 7th IEEE International Symposium on High Assurance Systems Engineering*, page 171, Washington, DC, USA, 2002. IEEE Computer Society.

[22] W. T. Tsai, X. Zhou, Y. Chen, and X. Bai. On testing and evaluating service-oriented software. *Computer*, 41(8):40–46, 2008.

[23] J. Vincent, G. King, P. Lay, and J. Kinghorn. Principles of built-in-test for run-time-testability in component-based software systems. *Software Quality Control*, 10(2):115–133, 2002.