Improving Testing of Enterprise Systems by Model-based Testing on Graphical User Interfaces

Sebastian Wieczorek and Alin Stefanescu
SAP Research
Darmstadt, Germany
{*name.surname*}@*sap.com*

***Abstract*—Software development and testing of Enterprise Resource Planning (ERP) systems demands dedicated methods to tackle its special features. As manual testing is not able to systematically test ERP systems due to the involved complexity, an efficient testing approach should be automatic. Since the underlying business processes of enterprise systems are realized at the level of the user interface (UI), system level testing is the dominant testing approach. The recent architectural shift to service-based enterprise systems demands to apply black-box testing techniques. Model-based testing is a method that enables a high degree of automation for black-box testing, but the current research and practice usually does not address the UI. In this paper we describe the state of the art and state of the practice in order to motivate further research activities in this area.**

***Keywords*-Model-based Testing; Enterprise Systems; Service-oriented Architecture; UI Testing; System-level Testing**

I. Introduction

Enterprise Resource Planning (ERP) software [1] supports business processes for whole companies. Such software systems are typically very large. SAP R/3 software for instance consists of more than 250 million lines of code [2]. ERP software integrates many organizational parts and functions into one logical software system, posing unique challenges to software development and testing [3]. Enterprise Service Oriented Architecture (Enterprise SOA) [4] is regarded as the next evolutionary step to cope with the software complexity of ERP systems where monolithic approaches are not applicable anymore. In Enterprise SOA, an application platform consists of independent business components exhibiting enterprise services that can be composed individually to form customized business processes.

Service-oriented architectures (SOA) are gaining pace towards becoming mainstream. Forrester studies and surveys [5], [6] of more than 2200 IT decision makers across North-America and Europe show that 2/3 of the companies expect to be using SOA by the end of 2009 while 60% of those currently using it are expanding their usage. Leading firms now use SOA on more than 50% of their solution delivery projects. SAP, a leading provider of business software, delivers SOA-enabled software, SOA methodology guidelines, and professional services [4]. Such a widespread SOA adoption implies that the quality assurance of the service-based systems becomes an activity of paramount importance, with a special focus on SOA testing.

While service unit testing is usually well researched [7]–[10] and consistently deployed in practice, other testing activities pose several new challenges [11], [12]. The difficulties to be overcome are due to the heterogeneity, high distributivity, dynamicity, and loose coupling of the service-based systems. These complexity properties are taking their toll on the testing process. A model-driven approach to SOA integration helps to address such challenges, as it allows for a general solution, applying state-of-the-art tools and techniques for formal reasoning about service models [10] and model-based testing (MBT) [13], [14].

In this paper we explain, why model-based GUI testing is an interesting but little covered research subject, especially considering the market relevance of enterprise software. In Section II we describe that the necessary application of the SOA paradigms for enterprise software systems implies that black-box testing techniques have to be used. In Section III we explain the relevance of testing SOA applications at system level. In Section IV we laid out that current test automation in the industry only takes place for the test execution and that any applicable testing approach for enterprise software has to be based on the system's user interface. In Section V we mention some related work on applying MBT on GUI systems. Section VI describes our plans on investing some more research in the area and Section VII concludes the paper.

II. Current SOA Testing Challenges

Since the most modern enterprise systems have SOA as the underlying paradigm, we must first understand the challenges in the developement and testing of these systems.

When new programming paradigms, such as the ones associated with SOA, are emerging, it naturally arouses the question whether there is a need for new testing methods or whether existing approaches can be adapted. More concrete, it has to be determined whether approaches and techniques, developed for traditional monolithic systems, distributed systems, component-based systems, and web applications can be adapted to service-based systems. In order to provide

an answer, the particularities and challenges of SOA testing have to be analyzed.

The authors of [11] identified the following key distinguishing factors for SOA that generate unique challenges for the testing activities:

- *Lack of code access.* For users, services are just interfaces as they neither have knowledge about the structure of the code nor the possibility to observe its execution. These limitations are preventing any form of white-box testing for users.
- *Dynamicity and adaptiveness.* For traditional systems, one is always able to determine at least the set of possible targets for a call [15]. This is not true for SOA where the work flow of abstract services might be bound to concrete services retrieved from one or more registries during execution.
- *Lack of control.* Services are deployed independently and might be updated without informing the consumers. Therefore service can unexpectedly change their behavior or miss service level agreements.
- *Lack of trust.* As decisions for choosing a certain service solely depend on information that the provider is publishing. Therefore there exists the risk that potential users are negatively influenced by providing them with incorrect or inaccurate information.
- *Cost of testing.* As test invocation by users may cause cost or other undesirable effects (e.g. an experience of a denial-of-service attack) on the provider side, extensive or repeated testing might not be feasible.

A slightly different approach has been taken in [16], where the challenges of SOA testing are clustered to the items *Stakeholder separation*, *Service integration*, *Service versioning and migration* and *Service binding and reconfiguration*. However, this regrouping neither leaves out nor introduces additional aspects.

Considering the challenges derived from [11], some general implications can be derived. It seems reasonable that the first three items - namely *lack of code access*, *dynamicity and adaptiveness*, and *lack of control* - are dealing with technical challenges while the following items - *lack of trust* and *cost of testing* - may have to be solved on the management level.

For addressing the management challenges group, it may be necessary to provide means of interaction between stakeholders, in order to share information and rights. Whether these means of interaction have to provide for anonymity, as it is presumed in [16] will be seen. However, the conservatism of major business companies and the consequent request for knowing and trusting business partners will probably lead to different solutions [1].

The technical chellenges group clearly requires that black-box testing techniques should be applied, as code access in a SOA environment is limited. Dynamicity and adaptiveness itself can only be achieved by providing detailed information about interfaces. Otherwise (in-)compatibilities cannot be detected and handled automatically. Therefore it can be assumed that MBT will be have a much greater impact in the testing process than in traditional industrial development setups, where modeling is carried out rather sporadically. The lack of control over parts of the system implies that tests will have to be carried out not only in the development of a service-based application, but also regularly after deployment. Therefore having automatic regression tests in place is a natural conclusion [17]. Moreover, the GUI testing might be in many case the only possibility of testing the functional and non-functional properties of a service-based system.

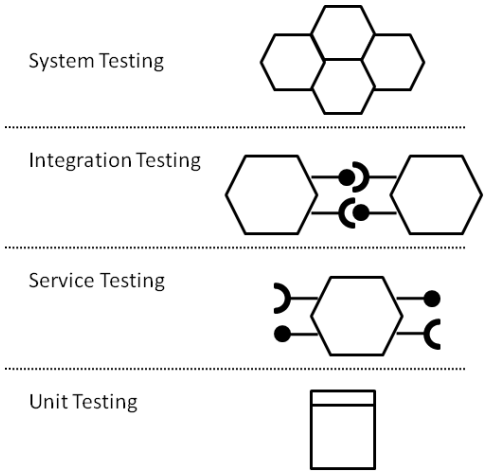

Figure 1. SOA testing layers as presented in [18]

III. The Enterprise SOA Testing Stack

After having discussed the general challenges of SOA testing, a closer look will be taken on the particular testing activities, in order to see which of them is affected in what way. In [13], the commonly used layered testing approach for component-based systems (CBS) has been described. As the general idea of partitioning applications into logical units is somehow similar to the SOA approach of encapsulating related functional units in a service, the definition of SOA testing layers can be done analogous to the one for CBSs. Consequently four distinct testing layers, illustrated in Figure 1, can be distinguished [18]. In the following each layer is shortly introduced.

Unit Testing: Unit testing is the best understood testing layer in research and practice. In contrast to all other testing layers, unit testing focuses on getting confidence in the functional correctness and hence in the correct implementation of the algorithms. As mentioned above, it deals with single software units in isolation. During unit testing the execution context of the software unit under test is mocked. Therefore, it can be carried out in SOA systems just like in any CBS implementation or any other software architecture, using all available tools and techniques.

Service Testing: Also service testing for SOA is analog to the testing of components in the CBS world to some extend. The general focus of service testing is less on the correct implementation of algorithms but on the integration of the functional units inside the (service) component and on the fulfillment of the contractual obligations of the component's interfaces. This is also conforming to the definition of testing layers [19], were it is argued that everything apart from unit testing is a form of integration testing.

Integration Testing: As mentioned before, the loose coupling of service components is one of the distinguishing factors of SOA. In contrast to the CBS approach, integration testing cannot rely on homogeneous components with tightly connected interfaces. Instead, the adaptability and distribution of SOA demands additional considerations for integration testing. Especially the effects of message racing and its implications have to be considered during system development and should be tested thoroughly [18]. Message racing in this context refers to situations where messages are not received in the same order as they are sent.

System Testing: In the SOA world, system testing can be defined analog to the classical definition for CBS, as comprising the fully integrated application, usually using its externally exposed interfaces. As the faultless interplay of the services can be assured on the integration testing level, in practice system testing is based on high-level usage scenarios and business requirements that have been defined by business analysts or customers. UI-based testing is therefore most appropriate to carry out the tests, as the system should be validated as a whole and only using access points that are available to the prospect user.

IV. A Glimpse in the State of the Practice

In the past, a significant effort has been put by the software industry to increase the efficiency of testing. Test automation in this context has been regarded as the most promising way. State of the practice of test automation for enterprise software is the automation of the test execution process, while the test design (the definition of abstract test cases and their concretization) is in general still a manual task.

At SAP, the transformation from abstract test cases to executable test scripts usually follows the keyword-driven testing principles. Keyword-driven testing (or action-word testing) uses action keywords in the test cases, in addition to data. Each action keyword corresponds to a fragment of a test script (the adapter code), which allows the test execution tool to translate a sequence of keywords and data values into executable tests [13].

The introduced keywords are for example realized on top of SAP's eCATT test script language [20]. To allow the highest possible reuse, they are oriented on the structure of the enterprise system itself, which is organized in so-called transactions. A transaction offers a set of methods to alter an enterprise's internal data in a consistent way. As most of the offered data modifications do not demand extensive computation, the transaction's functional logic is usually realized by the user interface and therefore can not be tested separately.

Therefore, implementing testing keywords is mostly done by utilizing capture/replay functionality, which is provided by most of the test automation tools. These tools are monitoring user interactions on the interface and producing a test script that can reproduce the execution of the recorded sequence of events. SAP's Test Workbench can be used to capture eCATT scripts, which further enables to give them more flexibility by exchanging concrete values with variables that can be initialized independently through the interface of the script. For complex transactions, a captured script can further be broken down to multiple scripts with lower complexity.

The test data used for the test runs on the SUT is usually very complex and additionally has to be compliant with existing master data and the actual system configurations [3]. Therefore, the current practice is to leverage the experience of the testers, who are asked to provide appropriate test data.

SAP further provides a tool called Test Data Migration Server (TDMS), which is able to derive consistent reference data from existing systems. It is also quite common that reference test data is provided by customers or internal departments, as additional information to the requirement specification. If these data samples are available, testers are able to choose the appropriate input for each test case from that source.

At SAP the test execution is controlled by the Test Workbench, where test plans (consisting of multiple test suites) are executed automatically and periodically in the case of regression tests. The results of the test runs are centrally reported, including different coverage criteria based on source code, model elements or requirements. Figure 2 shows, how eCATT automates the test execution, by having a global test scripts calling the referenced keyword scripts of each test step. The results of each test step are transferred to the next script using exporting and importing functions.

V. A Glimpse in the State of the Art on MBT for GUIs

Model-based testing has recently received a lot of attention both in the academic as well as the industrial communities, with a couple of books published [13], [14], [21] and several dedicated workshops (A-MOST, MBT, MOTES, MOTIP etc.). Moreover, there is already a market with several MBT commercial tools (like Qtronic from Conformiq, TestDesigner from Smartesting, and SpecExplorer from Microsoft). However, most of the tools and methods work for funtional testing during the software development phase and very few methods are available for the system level testing and especially UI testing. Some reasons for

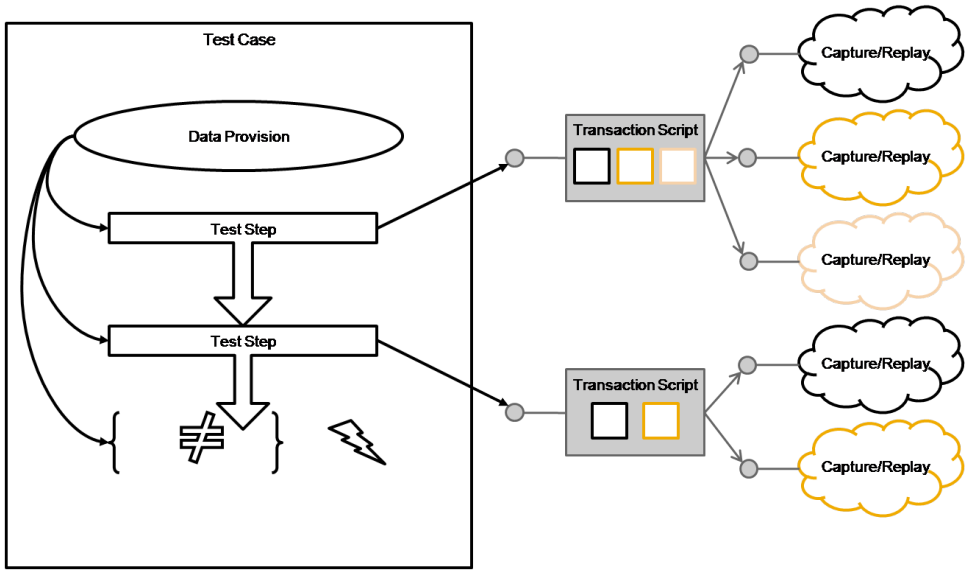

Figure 2. Industrial reuse concept of capture/replay scripts

that are the lack of precise models at the UI level based on the business processes, the large number of possible states and user events in modern GUIs and the UI dynamicity and runtime adaptation. One of the most complete reference on the research on MBT for GUI systems is [22]. This paper proposes a general event-flow model for GUI systems that can be used on the existing work of testcase generation, test oracle creation and regression testing. The work of [23] proposes annotated UML activity diagrams as modeling environement for GUIs. On the industrial side of research, [24] showed the difficulties of using the AGEDIS tools on a real GUI testing systems, whereas [25] shows how to use the Microsoft tool SpecExplorer for GUI testing.

VI. Planned Activities

As we have seen in the previous two sections, there is more need for research in order to successfully apply MBT to GUIs in an industrial setting. Figure 3 depicts the envisioned system testing approach that defines the research plan. As explained in Section III, system testing is carried out when the whole system (or at least major parts of it) is developed and test ready. In the following the individual steps of such approach are described. The development of the system or the adjustment of an already existing solution is left out of this description. It should happen previous or in parallel with the second step.

1) Members of consulting, key customers and development architects are deriving business process models for a new product or feature or customer implementation according to the market's or customer requirements and based on SAP's expertise on industrial best practices. Such processes can be modeled using the Business Process Modeling (BPM) product of SAP NetWeaver and then the UI elements and their connections can be modeled with the Visual Composer tool - see Figure 4 for two screenshots.
2) The created content, which effectively describes the usage scenarios of the new functionality is used to generate test model skeletons. By utilizing model transformation techniques this step should be automatic.
3) The test models are afterwards enhanced by test engineers, such that they reflect previously defined test goals and acknowledge the specifics of the concrete software architecture.
4) From the test models, abstract test suites are derived automatically, using model-based testing techniques.
5) The abstract test suites are optimized according to industrial best practices (e.g. minimizing test case length while preserving test coverage). After further concretizations, the optimized suite is executed automatically on the user interface of the system under test.

VII. Conclusion

In this paper we tried to motivate and draw some guiding lines of research in the area of system testing for enterprise systems, with a focus on GUI testing. The research will aim at improving the state of the art and state of the practice by using MBT techniques. This future work will leverage the existing work in GUI testing and MBT and try to find solutions that overcome the new challenges deriving from the new SOA paradigm. One key success element in this endeavor will be the modeling part that will have to address the complex test data, existing models for business processes and UI composition, and dynamic aspects of UI

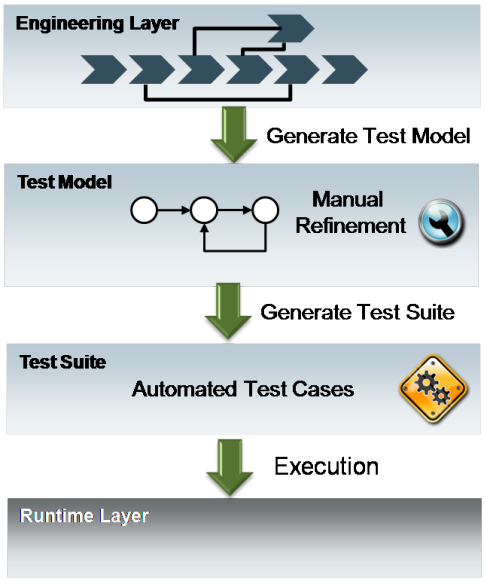

Figure 3. Envisioned testing process

at runtime. This research will contribute to a more reliable and automatically testable SOA enterprise applications.

ACKNOWLEDGMENT

This work was partially supported by the EC-funded projects Modelplex[1] and Deploy[2] (grants no. 034081 and 214158).

REFERENCES

[1] D. E. O'Leary, *Enterprise Resource Planning systems. Systems, life cycle, electronic commerce and risk*. Cambridge University Press, 2000.

[2] G. Pike, "Supporting business innovation while reducing technology risk," SAP AG, Tech. Rep., 2006. [Online]. Available: http://www.sap.com/services/programs/pdf/BWP_Supporting_Business_Innovation.pdf

[3] S. Wieczorek, A. Stefanescu, and I. Schieferdecker, "Test data provision for ERP systems," in *Proc. of Int. Conf. on Software Testing (ICST'08)*. IEEE Computer Society, 2008, pp. 396–403.

[4] D. Woods and T. Mattern, *Enterprise SOA - Designing IT for Business Innovation*. O'Reilly, 2006.

[5] Forrester, "Enterprise and SMB software survey, North America And Europe, Q4 2008," Forrester Research, Business Data Service Survey, 2008.

[6] R. Heffner, "Across all vertical industry groups, the majority of SOA users are expanding its use," Forrester Research, Research Report, May 2009.

[1]http://www.modelplex-ist.org
[2]http://www.deploy-project.eu

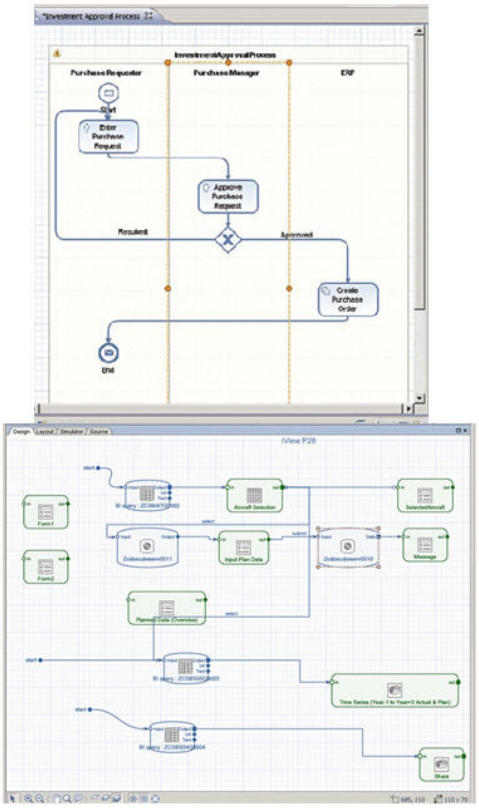

Figure 4. Example of models for business processes and UI composition

[7] C. Bartolini, A. Bertolino, E. Marchetti, and A. Polini, "Towards automated WSDL-based testing of web services," in *Intern. Conf. on Service-oriented Computing (ICSOC'08)*, ser. Lecture Notes in Computer Science, vol. 5364, 2008, pp. 524–529.

[8] W.-T. Tsai, Y. Chen, R. A. Paul, H. Huang, X. Zhou, and X. Wei, "Adaptive testing, oracle generation, and test case ranking for web services," in *29th Int. Computer Software and Applications Conference (COMPSAC'05)*. IEEE Computer Society, 2005, pp. 101–106.

[9] J. Offutt and W. Xu, "Generating test cases for web services using data perturbation," *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 5, pp. 1–10, 2004.

[10] L. Baresi and E. Di Nitto, *Test and Analysis of Web Services*. Springer, 2007.

[11] G. Canfora and M. D. Penta, "Service-oriented architectures testing: A survey," in *Software Engineering: International Summer Schools, ISSSE 2006-2008, Revised Tutorial Lectures*. Springer-Verlag, 2009, pp. 78–105.

[12] A. Barbir, C. Hobbs, E. Bertino, F. Hirsch, and L. Martino, "Challenges of testing web services and security in SOA implementations," in *Test and Analysis of Web Services*. Springer, 2007, pp. 395–440.

[13] M. Utting and B. Legeard, *Practical model-based testing, a tools approach*. Morgan Kaufmann, 2007.

[14] P. Baker, Z. R. Dai, J. J. Grabowski, Ø. Haugen, I. Schieferdecker, and C. Williams, *Model-Driven Testing: Using the UML Testing Profile*. Springer, 2008.

[15] A. Milanova, A. Rountev, and B. Ryder, "Parameterized object sensitivity for points-to analysis for Java," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 14, no. 1, pp. 1–41, 2005.

[16] M. Greiler, H.-G. Gross, and K. A. Nasr, "Runtime integration and testing for highly dynamic service oriented ict solutions," in *Proc. of Testing: Academic & Industrial Conference - Practice and research techniques (TAICPART'09)*. IEEE Computer Society, 2009.

[17] M. Acharya, A. Kulkarni, R. Kuppili, R. Mani, N. More, S. Narayanan, P. Patel, K. Schuelke, and S. Subramanian, "SOA in the real world - experiences," in *Service-Oriented Computing (ICSOC)*, vol. 3826, 2005, pp. 437–449.

[18] S. Wieczorek and A. Stefanescu, "Service integration: A soft spot in the SOA testing stack," in *To appear in Proceedings of the 5th Central and Eastern European Software Engineering Conference in Russia (CEE-SECR'09)*. IEEE Computer Society, 2009.

[19] S. Ali, L. C. Briand, M. J.-U. Rehman, H. Asghar, M. Z. Z. Iqbal, and A. Nadeem, "A state-based approach to integration testing based on UML models," *Information & Software Technology*, vol. 49, no. 11-12, pp. 1087–1106, 2007.

[20] M. Helfen, M. Lauer, and H. M. Trauthwein, *Testing SAP Solutions*. SAP Press, 2007.

[21] J. Jacky, M. Veanes, C. Campbell, and W. Schulte, *Model-based Software Testing and Analysis with C#*. Cambridge University Press, 2008.

[22] A. M. Memon, "An event-flow model of GUI-based applications for testing," *Softw. Test., Verif. Reliab.*, vol. 17, no. 3, pp. 137–157, 2007.

[23] M. Vieira, J. Leduc, B. Hasling, R. Subramanyan, and J. Kazmeier, "Automation of GUI testing using a model-driven approach," in *AST '06: Proceedings of the 2006 international workshop on Automation of software test*. New York, NY, USA: ACM, 2006, pp. 9–14.

[24] I. Craggs, M. Sardis, and T. Heuillard, "AGEDIS case studies: Model-based testing in industry," in *Proceedings of the 1st European Conference on Model Driven Software Engineering*. Imbus AG, 2003, pp. 129–132, online at: http://www.agedis.de/documents/AGEDIS in Industry.PDF.

[25] A. Paiva, J. C. P. Faria, N. Tillmann, and R. F. A. M. Vidal, "A model-to-implementation mapping tool for automated model-based GUI testing," in *ICFEM'05*, ser. Lecture Notes in Computer Science, vol. 3785. Springer, 2005, pp. 450–464.