Demonstration of an Automated Integrated Test Environment for Web-based Applications

Tiziana Margaria[1,2], Oliver Niese[2], and Bernhard Steffen[2]

[1] METAFrame Technologies GmbH, Dortmund, Germany
`TMargaria@METAFrame.de`
[2] Chair of Programming Systems, University of Dortmund, Germany
`{Tiziana.Margaria, Oliver.Niese, Steffen}@cs.uni-dortmund.de`

1 Introduction

The state of the art of test tools for web applications is still dominated by static approaches, focussing on *a posteriori* structure reconstruction and interpretation [9, 3] rather than on functional aspects directly related to the applications design. As it has happened in the areas of telecommunications and CTI, evolution in this direction is however necessary in order to test applications also for their intention, rather than just for environmental accidents.

We demonstrate a methodology capable of granting adequate coverage of functional testing, yet still easy to use. It bases on the experience we gained in system level test of Computer Telephony Integrated applications, where our coordination-based coarse grain approach has been successfully applied ([5]). Central properties of the test environment are

– the capability of handling distributed testing of web applications
– the capability of proving properties about the test cases, done via model checking, that ensure correctness of the test cases wrt. desirable criteria.

We show in the demonstration how the approach works in practice, by testing complex role-based functional features of the Online Conference Service [2], a cooperative work management system that supports the web-based organization of scientific conferences.

2 Dealing with Web-Based Applications

Modern *web-based applications* are complex multitiered, distributed applications that typically run on heterogeneous platforms. Their correct operation increasingly depends on the interoperability of single software modules, rather than just on their intrinsic algorithmic correctness. This has a huge impact on the test and validation requirements: it is increasingly unrealistic to restrict the consideration to single units, e.g. the client side only, since complex subsystems affect each other. Instead, scalable, integrated test methodologies are needed that can handle the application in a whole.

Figure 1 (in Appendix) sketches a typical architecture of a web-based application. The browser, located on a client, plays the role of the "classical" application GUI and interacts with a *Webserver*. The Webserver itself communicates with an *Application Server* which builds dynamically the requested web pages through an interaction with a *database* and (several) *Back-end services*. Often staging servers, firewall solutions, load balancing and redundant architectures increase the speed and availability of the offered application, and increase the complexity of the considered test setting. So test methods are needed that can deal with this kind of complex distributed system: it is obviously not sufficient to test the just the user interface, it is rather of central importance to consider the server side as well (i.e. the web-server, application-server and the back-end services). This task exceeds today's commercial internet-testing tools, as marketed e.g. by Compuware [1] or Radview [7]. An adequate approach should on the one hand allow an easy, graphical design of test cases and on the other hand the execution of these complex test cases through a coordination of different, heterogeneous test tools. These test tools locally monitor and steer the behavior of the software on the different clients/servers, which are seen as units under test.

Our integrated test environment addresses the web-based, distributed application in a whole and covers mostly the global functional aspects, along the intuitive description of the approach of [6]. To test this kind of complex systems, a *Test Coordinator* drives the generation, execution, evaluation, and management of the system-level tests in the highly heterogeneous landscape, cf. Fig. 1(in Appendix). The coordination layer introduces the required flexibilization of the overall architecture of the test environment: it is a modular and open environment, so that diverse tools and units under test can be added at need. The heart of the environment is the Test Coordinator tool, built on top of METAFrame's *Agent Building Center* [10], which is a generic and flexible workflow management system, cf. Fig. 2. The *ABC* provides in particular support for the design, verification and execution of workflows.

3 Case Study: Testing the Online Conference Service

We demonstrate the key features of the *ITE* on a real life example: the test of the Online Conference Service [2], a complex web-service that proactively helps authors, Program Committee chairs, and Program Committee members to cooperate efficiently during their collaborative handling of the composition of a conference program. The application provides a timely, transparent, and secure handling of the papers and of the related submission, review, report and decision management process. The online service includes in particular a role based access and rights management feature that is reconfigurable online, which accounts for a high flexibility and administration comfort, but which is responsible for potentially disruptive behavior in connection with sensitive information. Because of the complexity of the underlying workflows (and of course therefore the complexity of the application itself) intensive testing is clearly unavoidable.

The test process supported by the *ITE* is organized in the following main steps: *identification* of generic test blocks, *design*, *verification* and *execution* of test cases.

Identification of Generic and Reusable Test Blocks The first task is to identify generic, test actions which allow us to control the online service via a browser and also to validate certain properties of the service state (e.g. by checking the current state of the GUI or by checking the entries of an underlying database). This is consistent with the coarse grained design approach to the services presented in [11]. For each action a test block is prepared: a name and a class characterizing the block are specified and a set of formal parameters is defined to enable a more general usage of the block. In this way, for the *Web-based application* to be tested a library of test blocks has been incrementally defined. It includes test blocks representing and implementing, e.g.

Common actions for the initialization of test tools, system components, test cases and general reporting functions,

Common internet related actions e.g. starting/initialization of a browser, following a link,

Application-specific actions: miscellaneous actions to operate a specific application via its graphical user interface, e.g., log-on/log-off of a user, or checking labels of GUI-elements.

Graphical Design of Test Cases The design of test cases consists in the behavior-oriented combination of test blocks. This combination is done graphically, i.e., icons representing test blocks are graphically stuck together to yield test graph structures that embody the test behavior in terms of control.

The test case shown in Fig. 4 tests wether an article, submitted by an author, is delegated correctly to a member of the program committee. From the submission to the start of the review process of an article at least three different roles are involved: the *Author* himself, the Conference Chair (*PCChair*), who must delegate the article to a member of the program committee, and finally the members of the program committee (*PCMember*), who are responsible for the reviews. So we need at least three concrete users to run this test, whereby each user is associated with a distinct role and located at a distinct client machine. At the beginning of the test case for every user a browser must be started to perform the required tasks. The single steps to be executed are here encapsulated in a *macro*, which can be used as reusable atomic test actions in the design of new test cases.

Verification of the Test Cases In our environment, the design of test cases is constantly accompanied by online verification of the global correctness and consistency of the test cases' control flow logic [4]. During the test case design, vital properties concerning the usage of parameters (local properties) and the interplay between the stimuli and verification actions of a test case (global properties, expressed in a user-friendly specification language based on SLTL, the Semantic Linear-time Temporal Logic of [10]) can be verified. Test cases conflicting with the constraints and consistency conditions of the intended system are thus immediately detected.

Systems developed with the Agent Building Center, like the Online Conference Service, enjoy in this respect a double bonus: the same functional constraints enforced at design time (as detailed for this particular service e.g. in [11, 2]) can now be reused to validate the test cases, ensuring consistency of the test purposes with the original system specification.

Typical test-phase specific additional constraints for the test of *Web-based applications* refer e.g. to the resource management at test execution time: one can ensure that all used resources are correctly released after the execution of the test case[1], and this independently of the tests outcome. If the model checker detects an inconsistency, a plain text explanation of the violated constraint appears. In addition, test blocks violating a local property as well as paths violating a global property are marked.

Execution of the Test Cases and Reporting Test cases can be executed immediately in the *Test Coordinator* by means of *ABC*'s tracer. Starting at the initial test block of a test graph the tracer proceeds in interpreter modus from test block to test block. Actions represented by a test block are performed, i.e., stimuli and inspection requests are sent to the corresponding system's component, responses are received, evaluated, and the evaluation result is used to select the relevant next test block.

Figure 4 illustrates these features on a concrete snapshot of a test case execution: here, an *Author* submits an article that should be reviewed by a member of the program committee. Figure 3 shows the delegation-based interaction between the involved system components. Here we see that, like in telecommunication system testing, the interaction between Test Coordinator and single components of the system under test is always mediated by an instance of the dedicated test tool (in this case, Rational Robot) responsible for handling the stimulation and observation of this particular component (in this case, a client browsers GUI). Even this relatively simple scenario shows the potential complexity of the management and execution of such distributed tests.

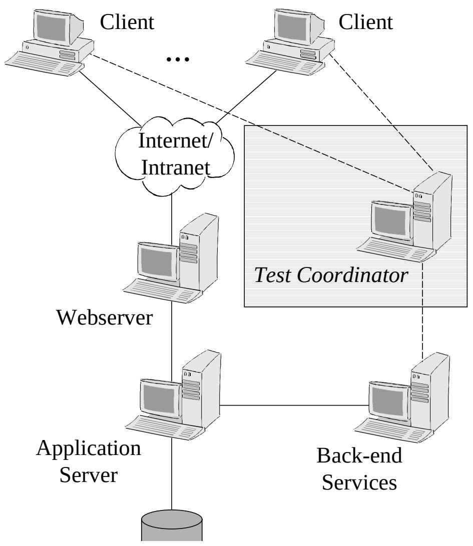
References

1. Compuware Inc. http://www.compuware.com.
2. B. Lindner, T. Margaria, and B. Steffen. Ein personalisierter Internetdienst für wissenschaftliche Begutachtungsprozesse. In *GI-VOI-BITKOM-OCG-TeleTrusT Konferenz on "Elektronische Geschäftsprozesse"*(eBusiness Processes), Universität Klagenfurt, September 2001.
3. C.H. Liu, D.C. Kung, P. Hsia, and C.T. Hsu. Structural testing of web applications. In *Proc. of Int. Symposium on software reliability engineering (ISSRE 2000)*, pp. 84–96, 2000.

[1] E.g. every log-in for an user must be followed by a log-out for this user.

4. O. Niese, B. Steffen, T. Margaria, A. Hagerer, G. Brune, and H. Ide. Library-based design and consistency checks of system-level industrial test cases. *Proc. FASE 2001)*, LNCS N. 2029, pp. 233–248. Springer Verlag, 2001.
5. O. Niese, T. Margaria, A. Hagerer, B. Steffen, G. Brune, and H. Ide. Automated regression testing of CTI-systems. In *Proc. of IEEE European Test Workshop (ETW 2001)*, 2001.
6. O. Niese, T. Margaria, B. Steffen. Automated functional testing of web-based applications. Accepted for publication in 5th Intern. Quality Week Europe 2002, Brussles, 2002.
7. Radview Software. http://www.radview.com.
8. Rational, Inc. The Rational Suite description http://www.rational.com.
9. F. Ricca and P. Tonella. Building a tool for the analysis and testing of web applications: Problems and solutions. *Proc. TACAS 2001*, LNCS N. 2031, pp. 372–388. Springer Verlag, 2001.
10. B. Steffen and T. Margaria. *METAFrame in Practice: Design of Intelligent Network Services*, LNCS N. 1710, pp. 390–415. Springer Verlag, 1999.
11. B. Steffen, T. Margaria, and V. Braun. Coarse-granular model checking in practice. Proc. 8th International SPIN Workshop, LNCS N. 2057, pp. 304–312. Springer Verlag, 2001.

A Illustrative Material



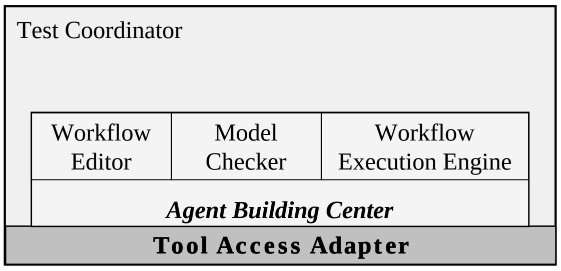


Fig. 2. Overview of the Test Coordinator

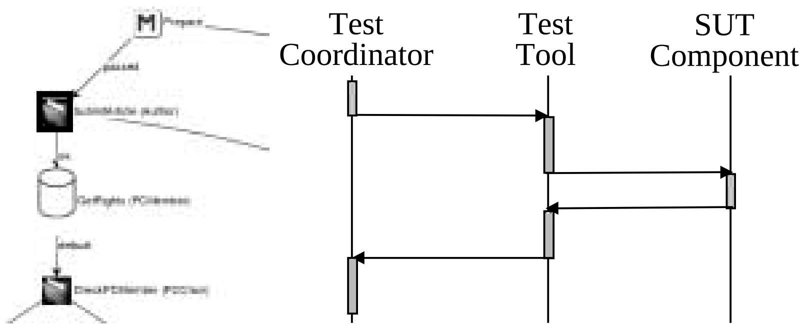


Fig. 3. Test Execution

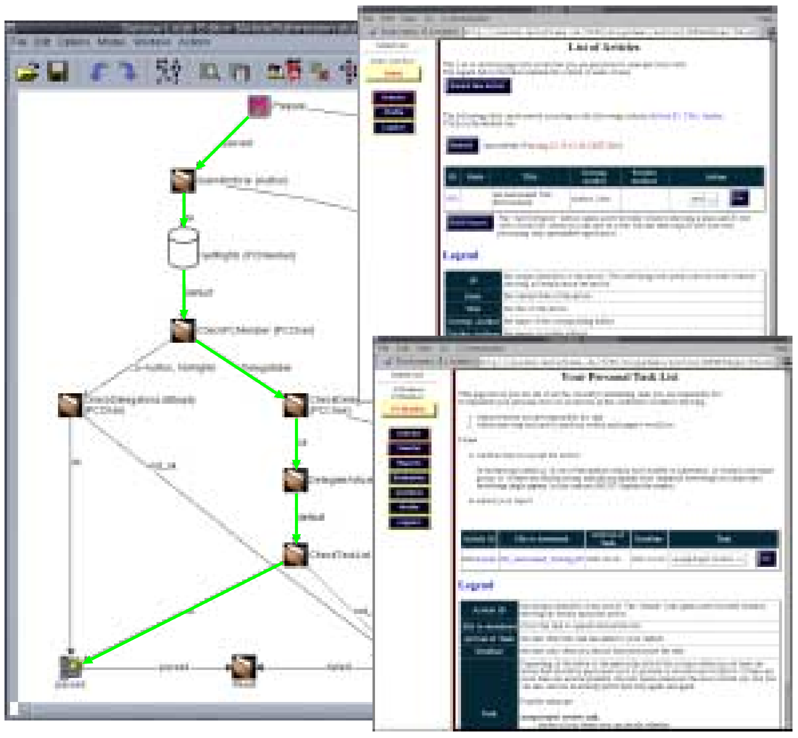


Fig. 4. Test Execution: A Successful Execution Path and the Users Views