Demonstration of an Automated Integrated Testing Environment for CTI Systems

Oliver Niese[1], Markus Nagelmann[1], Andreas Hagerer[1], Klaus Kolodziejczyk-Strunck[2], Werner Goerigk[3], Andrei Erochok[3], and Bernhard Hammelmann[3]

[1] METAFrame Technologies GmbH, Dortmund, Germany
{ONiese, MNagelmann, AHagerer}@METAFrame.de
[2] HeraKom GmbH, Essen, Germany
klausk@herakom.de
[3] Siemens AG, Witten, Germany
{Werner.Goerigk, Andrei.Erochok, Bernhard.Hammelmann}@wit.siemens.de

1 An Integrated Testing Environment Based on ABC

We demonstrate the *Integrated Testing Environment*, ITE, an environment for automated and integrated testing at system level. A companion paper [4] describes the problem of systems integrated testing in its conceptual application modelling focus. Here and in [5], the concept's implementation is exemplified in case of automated integrated testing of a Computer Telephony Integration (CTI) system. The CTI system consists of a switch, an automatic call distributor, and a suite of applications forming a call center and supporting the tasks of a center's human agent, e.g., applications that enable an agent to log-on/log-off at the call distributor, or to initiate conference calls with other agents.

In the ITE, each hardware and software component of the CTI system is controlled by its own test tool, e.g., a proprietary hardware tracer for the switch or a GUI test tool such as *Rational SQA Robot* [6] for the applications. Coordinating which action has to be performed by which test tool is under responsibility of the *Test Coordinator*, a tool built on top of METAFrame Technologies' *Agent Building Center* (ABC) [2], a general-purpose environment for specification and verification of complex workflows.

As outlined in the following sections, the ITE supports the design of test cases including the verification of their consistency, the interactive combination of test cases resulting in test scenarios, and the execution of test cases and test scenarios in the heterogenous processing environment of CTI systems.

2 Design Support Features

System testing is characterized by focussing on inter-components cooperation. For the design of appropriate system-level test cases it is necessary to know what features the system provides, how to operate the system in order to stimulate a feature, and how to determine if features work. This information is gathered and after identification of the system's controllable and observable interfaces it is transformed into a set of stimuli (inputs) and verification actions (inspection of outputs, investigation of components' states). 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 CTI system to be tested a library of test blocks has been issued that includes test blocks representing and implementing, e.g.

Common actions. Initialization of test tools, system components, test cases and general reporting functions,

Switch-specific actions. Initialization of switches of different extensions,

Call-related actions. Initiation and pick up of calls via a PBX-network or a local switch,

CTI application-related actions. Miscellaneous actions to operate a CTI application via its graphical user interface, e.g., log-on/log-off of an agent, establish a conference party, initiate a call via a GUI, or check labels of GUI-elements.

The library of test blocks grows dynamically whenever new actions are made available.
The design of test cases consists in the behaviour-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 behaviour in terms of control.

3 Verification Support Features

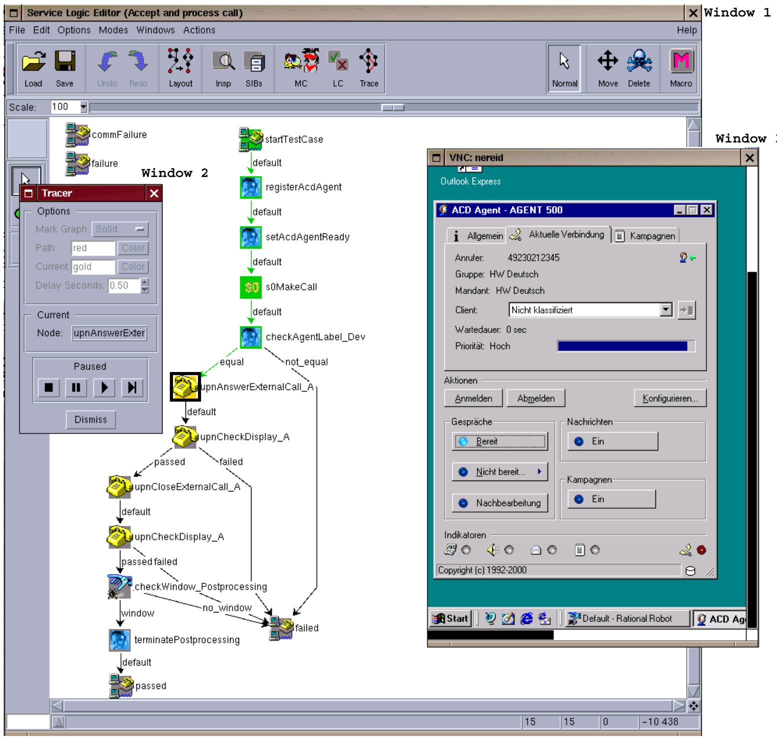
As explained thoroughly in [4], design of test cases is constantly accompanied by online verification of the global correctness and consistency of the test cases' control flow logic. During the design phase, vital properties concerning the usage of parameters (local properties) and concerning the interplay between the stimuli and verification actions of a test case (global properties) can be verified. Design decisions that conflict with the constraints and consistency conditions of the intended system are thus immediately detected.

Local properties specified imperatively are used to check that the parameters are correctly set. Global properties concerning the interactions between arbitrarily distant test blocks of a test graph are expressed in a user-friendly specification language based on the Semantic Linear-time Temporal Logic [3], and are gathered in a constraint library accessed by the environment's model checker during verification. Basic constraints have been formulated for the CTI system, e.g., if a test graph includes an action "hook-on", this must be preceded by an action "hook-off".

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.

4 Execution Support Features

In the ITE, test cases can be executed immediately by means of ABC's tracer. Starting at a dedicated test block of a test graph the tracer proceeds from test block to test block. The 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 one of the possibilities to pass control flow to a succeeding test block.



Scale: 100

Mark Graph: Solid
Path: red Color
Current: gold Color
Delay Seconds: 0.50
Node: upnAnswerExter

Anrufer: 49230212345
Gruppe: HW Deutsch
Mandant: HW Deutsch
Client: Nicht klassifiziert
Wartedauer: 0 sec
Priorität: Hoch

Copyright (c) 1992-2000

15 15 0 -10 438

The screen shot illustrates this. The system-under-test is a call center application, in this case a client-server CTI application called "ACD Agent" which runs on different computers than the *Test Coordinator*. In this test we emulate a human call center agent with identifier *AGENT 500* and handle some actions via the agent's GUI of the PC application. The test case graph is shown in Window 1. The Tracer-window (Window 2) controls the execution of the test case. The test block executed last is highlighted in the test graph window. Window 3 shows the agent's computer screen.

The implementation of this execution scheme requires two activities during set-up of the ITE. First, the actions referenced via test blocks have to be implemented by means of test tools. This task is performed by test engineers which are familiar with test tools, their handling and programming. For each action, the test engineers has to specify instructions to be executed by the test tool determined to support the specific action, e.g., via recording GUI-activities. Second, specific tracer code that is assigned to the action's test block and that will be executed by the tracer has to be developed. Experience with the CTI system shows that this code can be generated automatically for most actions. Manual development is necessary only if the test block shall initiate the execution of multiple actions in order to meet real-time requirements or if more complex evaluation of information about a component's state or reaction is required.

To communicate with different test tools, a flexible CORBA-based interface has been designed for the ITE. This interface comprises basic methods which all test tools have to support as well as special features of tools added by means of specializations. In the *Test Coordinator*, these methods are wrapped in a uniform way and are provided for implementation of tracer code in form of adapters. The extensibility of the ITE by additional adapters for other test tools is the key of the approach. On the test tool's side, the interface functionality has to be provided either by implementing a plug-in including a CORBA object request broker or by implementing a separate server process that communicates with the test tool via a dedicated interface (e.g., COM/DCOM).

At the moment, two test tools have been made accessible in the ITE: *Rational SQA Robot* [6] for driving an application's GUI and a proprietary tool *Hipermon* developed by HeraKom GmbH. This tool controls a device simulator that communicates with the switch and that is able to accurately emulate calls.

Finally, when executing a test graph, a detailed protocol is prepared. For each test block the tracer executes, all relevant data (its execution time, its name, the version of the files associated with the test block, the block's parameter values, and the processed data) are written to the protocol.

References

1. S. Gladstone: *Testing Computer Telephony Systems and Networks*, Telecom Books, 1996.
2. B. Steffen, T. Margaria, V. Braun, N. Kalt: *Hierarchical Service Definition*, Annual Review of Communic., Vol. 51, Int. Engineering Consortium, Chicago, 1997, pp.847-856.
3. T. Margaria, B. Steffen: *Backtracking-free Design Planning by Automatic Synthesis in METAFrame* Proc. FASE'98, Int. Conf. on Fundamental Aspects of Software Engineering, Lisbon, LNCS 1382, Springer Verlag, 1998, pp.188-204.
4. O. Niese, B. Steffen, T. Margaria, A. Hagerer, G. Brune, H.-D. Ide: *Library-based Design and Consistency Checking of System-level Industrial Test Cases*, FASE 2001, this volume.
5. O. Niese, T. Margaria, A. Hagerer, M. Nagelmann, B. Steffen, G. Brune, H.-D. Ide: *An Automated Testing Environment for CTI Systems Using Concepts for Specification and Verification of Workflows*, accepted for publication in Annual Review of Communic., Vol. 54, Int. Engineering Consortium, Chicago, 2000.
6. Rational, Inc.: *The Rational Suite description*, `http://www.rational.com/products`.