Testing of Interorganizational Workflows

Alexander Lechner

world-direct.at eBusiness solutions GmbH
a Telekom Austria company
Maximilianstrasse 2/B/1
A-6020 Innsbruck
alexander.lechner@world-direct.at

Abstract: This paper presents FLOWTEST a concept on testing interorganizational workflows on model-level. FLOWTEST focuses on building up test-models, deriving test-cases and executing tests on predefined workflows consisting of atomic web-services. The main goal of FLOWTEST is the realization of a test-suite to drive round-trip testing from code to model back to code level.

1 Introduction

Transforming interorganizational business-process from an organizational point of view into IT-managed execution is a new challenge for software development. Most of the time, fairly different IT-systems of quite unequal organizations must work on one "common process". The level of automation has a range from "supported" to "completely autonomous" without manual interaction. Regarding this process from the perception of software development, business-processes can be defined using some new techniques for a consolidated design of common processes like BPEL [AND+03], but comprehensive test-infrastructures and staging facilities to support further steps of software development are missing. Due to these facts this paper presents FLOWTEST - an industrial cooperation project between Telekom Austria and the University of Innsbruck. It's main goal is to provide an environment to perform tests of workflows on an abstract level. Therefore early defined workflows are raised on a model-level using UML class- and activity-diagrams for describing. "FCL" as a derivate of OCL is applied to add semantic information to these models and to define test-cases. Section 2 will outline a short example which highlights the need for testing interorganizational workflows, section 3 covers the testing-roundtrip supported by FLOWTEST and sections 4-6 focus on the main concepts of FLOWTEST including test-case specification.

2 Example

To point out the need for testing workflows the following real-world example presents a typical workflow out of the telecom world concerning a broadband access ordering pro-

cess. For demonstration purposes in this paper it was very simplified.

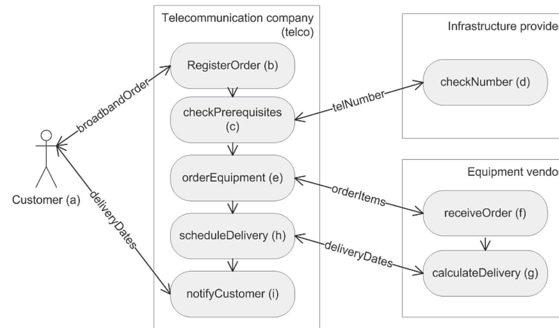

Figure 1: Broadband ordering process (simplified)

The workflow is initiated by a customer (a) order which is entered into the telco‘s (provider) ordering-systems (b). The provider itself uses the services of an infrastructure service provider (ISP), whose web-service “checkNumber” (d) is called to check the customer’s line for broadband capability (d). If the customer’s line is broadband-ready the necessary equipment has to be ordered (e) at an equipment vendor (EV) by calling his “receiveOrder” service (f). This function is followed by an internal calculation of delivery dates (g) which are responded to the telco’s scheduleDelivery interface (h) and handed on to the customer by telco’s “notifyCustomer” service. Assume that both contributors (ISP and EV) offer their service-interfaces defined as WSDL-files and the workflow was designed by all attendees in BPEL in advance. The telco’s systems are responsible for the execution and the monitoring of the process. To do so, they contain an engine which is capable of defining and executing BPEL-defined workflows.

3 Concept

This chapter presents the main concepts of the model-based workflow testing approach. The testing process of FLOWTEST is defined as a code-to-model-to-code sequence, because an executable workflow definition joint with XML documents (as the workflow’s payload) is lifted up to a model level to define the testing activity. The testing process is finished at code level with an “enhanced” BPEL workflow definition. As a BPEL defined workflow consists of atomic parts and their orchestration, bugs can occur in calling the involved web-services as well as in the workflow itself. Therefore a 2-layered testing concept is introduced: a) local testing of the involved partners (atomic components) in “system simulator” (see figure 2) and b) global testing of the workflow (orchestration of atomic components) in “flow tester” (see figure 2). A test sequence is initiated by loading WSDL-files to build up the system simulator (1). Test cases are generated either automatically or manually and are fired against the atomic web-services (2, 3) [BRE+04]. Results of this stage are used as inputs for the global execution model (flow-tester) (4) whose output would be an enhanced interface and/or flow-definition. Test-cases and test-execution results are stored in a test-database (5). Sections 3.1 and 3.3 will explain the activities in detail.

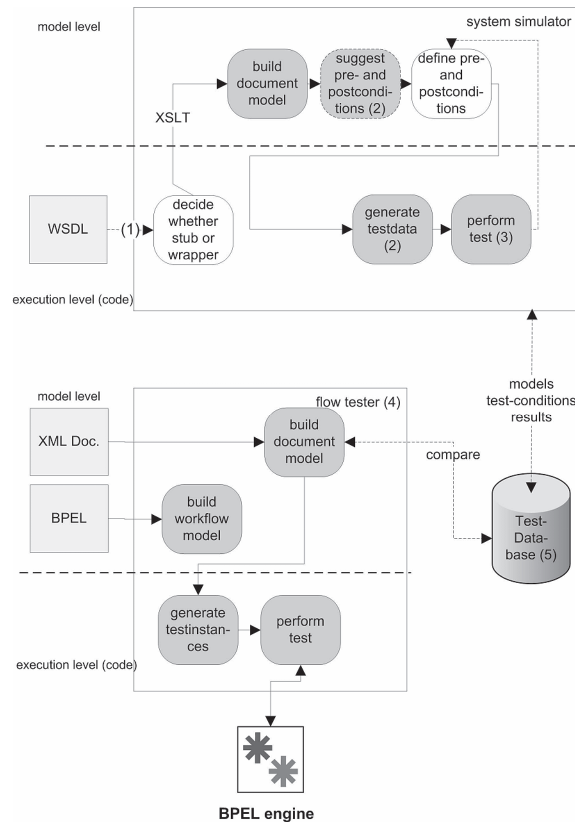

Figure 2: FLOWTEST main components

3.1 Local Testing

To carry out tests on the components of the workflow, the WSDL-files referenced in the global workflow definition, are loaded into the "system simulator". This section of the test-suite could act autonomously but - for better test results - the interaction with a test-agent (person) is recommended. A typical test-sequence proceeds as follows:

1. WSDL-files referenced in the BPEL-workflow are loaded. It can be decided by the user, whether the system simulator should produce stubs or wrap the remote functionality. This decision would heavily depend on the existence of testing and/or integration facilities at the web-service provider.
2. A document-model is set up according to the `<type>` and `<schema>` tags contained. This is realized by applying XSLT stylesheets to the WSDL-files which transform type- and operation-information from code to UML class-, attribute-, operation- and parameter information. As we can refer to WSDL-schema, XSLT-rules like described in appendix [XSLT] can be applied to named WSDL-elements.
3. Out of this document-model either the system can calculate and/or test-conditions are defined by manual user interaction (also see 3.2). The idea is that automated testing

(e.g. equivalence-class testing, boundary-values etc.) should be done by the system, while test-conditions based on semantic information should be provided by the user. Figure 3 denotes an example for system- and user-defined test-conditions.

4. Conditions no. 1, 2 and 3 are automatically set up by the system simulator, "learning" from the interface-definition of the web-service. The test execution will show, that nor no. 2 neither no. 3 will be accepted by the tested interface and produce a "not ok" output. Conditions no. 19 and 20 are manually entered by a tester (see chapter 3.2).

class: CheckNumber_Stub	client-type: stub	operation: string checkTelNumber(telNumber: string)
cond. no.	input	output
1	"01-12345"	"ok"
2	"324552"	"ok"
3	"$%7Htg+~"	"ok"
…		
19	pre: numberOk: telNumber.exclude ("$","/")	post: result = 'ok'
20	pre: numberNotOK: telNumber.startsWith('99')	post: result = 'not ok'

Figure 3: Test-case specification

5. A test is performed by applying system-generated test-values against user-defined pre-conditions (step 1). Some test-cases won't already match these criteria. In step 2 the enduring system-generated and user-defined test-cases are fired against the operation on behalf of the test client (wrapper or stub). Results of test-execution are joint to the test-cases (or conditions) and are stored in the test-database. Seeking the test-case results maybe new conditions can be derived by the tester [SNE+02].

3.2 Test-case specification

A test case is defined as a tuple consisting of three components: an operation call, an input- and an output-condition. Operation calls are derived from the methods provided by the atomic web-services involved in the workflow. This component is obligatory. Either an input or an output-value is compulsory for a valid test-case. In this manner a test-case defines a function $O=f(I)$ whereby f is the operation call on the input-value (I) producing an output O. Conditions are given for the function as a whole (expl. a)) or only for an in- or output condition (expl. b)). To provide efficient defining facilities for conditions "FCL" is introduced. "FCL" stands for "**F**lowTest**C**ondition**L**anguage" and is derived from OCL.

```
context: CheckNumber_Stub::checkTelNumber(p1: string): string
        pre: inputOk: p1 = "12345"
        post: resultOk: result = "ok"
```

This FCL-statement (expl. a)) defines a test-case on the "checkTelNumber"-operation of the generated CheckNumber_Stub class (in system simulator). It says that the input-value "12345" has to produce the output-value "ok" otherwise the test-case is negative.

```
context: CheckNumber_Stub
        pre: self.telNumber.includes(?-?)
```

This FCL-statement (expl. b)) performs a test on the attribute "telNumber" of Check-Number_Stub class. It says that this attribute always has to contain a hyphen ("-") to be correct.

3.3 Global testing

In this area, well-known methods of white-box testing can be applied. The FLOWTEST component "flow tester" takes XML-documents as input to perform tests the `<flow>` component of the BPEL-defined workflow (test object). A recommended test-sequence of flow tester:

1. Generate XML-instance documents. The main input for the generation of XML-instances is given by document definitions (XML-schemas) or the document model of the BPEL-workflow which is itself exposed as WDSL (see system simulator).
2. Run workflow on a BPEL-engine using the XML-instance documents as initial inputs. Therefore a BPEL-engine with an API must be used to provide external intervention into a running BPEL-instance. Arriving at an atomic web-service (BPEL: partnerlink) in executing the flow-component the actual document-model of the XML-instance is compared to the appropriate document-model of the partnerlink. This document model is known from system simulator and has already been enhanced by test-conditions. The comparison could result in an error when the XML-instance document-model doesn't fit the test-conditions defined for the partnerlink.
3. Flow tester records a visit log of all branches and conditions visited in the execution of the flow component [THA02]. In this way not tested sub-routines can be found (branch coverage test) [SPI+04]. As a test result of the testing-process flow tester reports necessary, additional fault handling (workflow positions) and branch coverage statistics.

4 Conclusion and further work

The preceding sections presented FlowTest which introduces a new concept for round-trip testing of interorganizational workflows. In further steps of research we will work on the integration of test-cases on model-level and on precising transformation capabilities of FLOWTEST. A special interest will be put on enhancing known standards like BPEL by test-results.

5 References and Related Work

[AND+03] T. Andrews, et.al.: Business Process Execution Language for Web Services V1.1: Spezifikation. http://www-106.ibm.com/developerworks/library/ws-bpel/, Mai 2003.

[BRE+04] R. Breu, M. Breu, M. Hafner, A. Novak: Web Service Engineering – Advancing a New Software Engineering Discipline. Accepted for ICWE 2005.

[SNE+02] H. Sneed, M. Winter: Testen objektorientierter Software, Hanser Verlag, München 2002.

[SOA05] S. Dustdar, St. Haslinger: Testing of Service Oriented Architectures. Distributed Systems Group, Vienna University of Technology, 2005.

[SPI+04] A. Spillner, T. Linz: Basiswissen Softwaretest. d.punkt Verlag, Heidelberg 2004.

[THA02] G. E. Thaller: Software-Test, Verifikation und Validation. 2., aktualisierte und erweiterte Auflage. Verlag Heinz Heise, Hannover 2002.

[XSLT] http://www.world-direct.at/flowtest/xsltExample.html