A NEW APPROACH TO MODEL A FORMALISED DESCRIPTION OF A COMMUNICATION SERVICE FOR THE PURPOSE OF FUNCTIONAL TESTING

Patrick Wacht, Thomas Eichelmann, Armin Lehmann, Woldemar Fuhrmann, Ulrich Trick and Bogdan Ghita

Research Group for Telecommunication Networks,
University of Applied Sciences Frankfurt/M., Germany

Centre for Security, Communications and Network Research,
University of Applied Sciences Plymouth, United Kingdom

University of Applied Sciences Darmstadt, Germany
wacht@e-technik.org

ABSTRACT

This paper presents a concept of how a service provider can verify that an implemented communication service meets the requirements of a customer. This requires functional tests which are derived from a finite state machine-based behaviour model being composed from predefined modular sub finite state machines. As the composition of these modular finite state machines to a behaviour model is done by retrieving information from the requirements specification or rather Service Description, the model reflects the business logic of the communication service. A case study of the modelling procedure is shown in this paper by means of an example service.

KEYWORDS

Functional Testing; Behaviour Model; Finite State Machine; Model-Based Testing

1. INTRODUCTION

The complexity of value-added communication services is ever increasing. In the telecommunication domain, a malfunctioning of services may be costly or even compromises the reputation of a specific service provider. Therefore, functional testing procedures have to be executed consequently before the delivering of the service to a customer, because the provider has to assure that the service is executed properly and does not affect other running services within the provider's service environment.

Functional testing is considered a sub-category of black-box testing and the construction of the test cases is solely done manually by a test developer from the information given in a specification which is supposed to define the behaviour of a system or rather service. In general, the test developer has to spend a significant amount of time on test case design, test data selection, and test evaluation because there are no adequate tools available to automate these tasks for testing of communication services. So, new mechanisms have to be evolved to help overcome this situation, thus increasing both efficiency and effectiveness of the testing process.

This paper which demonstrates concepts of the corresponding project ComGeneration [1] proposes a new approach to compose a so-called behaviour model from predefined building blocks which can be created by a test developer from the information he could retrieve from a certain requirements specification. Both the behaviour model and the predefined building blocks are finite state machines (FSM) whose paths represent possible message flows. To create the

behaviour model, the test developer will use a graphical editor to design the model itself and provide the test configuration and test data. The whole modelling procedure will be the focus of this paper.

The retrieving of test cases from the behaviour model and the generation and execution on the System under Test (SUT) will not be discussed. Further information on this purpose can be read in [2].

The remainder of this paper is structured as follows: Section 2 introduces the related research. In Section 3, the consistent concept of the approach is described. Section 4 contains a case study which introduces the development of a behaviour model by means of a service example. Finally, a conclusion is provided in Section 5.

2. RELATED WORKS

Among the existing researches, test case generation based on models – model-based testing (MBT) – describing the intended behaviour of a system is proposed by different authors. The MBT approach is used for various kinds of software.

Conformiq [3] describes tests by UML state diagrams, but the focus is not to describe the service from the view of the SUT, but from the view of the test components. A similar approach from Yuan et. al [4] describes test case generation from UML activity diagrams, but the main focus is about testing Web Service compositions with the help of TTCN-3. Gönczy et. al [5] address the testing of service infrastructure components against their specifications. They proposed a technique to synthesise a compact Petri net representation for the possible interactions between the service under test and the test environment. Concrete test cases can be defined by a sequence of controllable actions in this Petri net. Brucker et. al [6] report on their experience on how model-based descriptions can be used to derive tests for security policies. They were able to derive tests from models for stateless and statefull firewalls. Ali et. al [7] discuss closely related work to those of the author. In their approach, the use of UML 2.0 state machines, composed of several sub machines, is proposed. They implemented a model-based testing tool with the name TRUST (Transformation-based tool for Uml-baSed Testing) which is able to flatten the complex state machine and transform it into a test model. However, their approach still requires a lot of interactions with a user and therefore is not applicable for automated test execution as desired by the author's research project. Wieczorek at. al [8] developed an infrastructure in which complex software systems are described by the usage of model-driven engineering (MDE). The main goal of this project is to develop a standardised solution to improve the quality and productivity. Testing aspects are limited to model verification techniques and model-based simulations, allowing only the observation of the service's behaviour. Zhang Xiaoyan et al. [9] research generates test cases based on the OWL-S requirement model in order to test the interaction of several Web Services. Pretschner et. al [10] give a general overview about common approaches and challenges model-driven testing is facing. Tretmans et. al [11] highlight the benefits for a completely automated test support originating from the test code generation from a model, over the test execution to the analysis of the test results.

The introduced research activities have in common that they only support artefacts of the whole software testing process. Most of the concepts do not provide a consequent procedure from the requirements specification to the execution of tests and, furthermore, do not provide any predefined building blocks like the modular FSMs in the ComGeneration approach.

3. CONCEPT OVERVIEW

Before looking at the practical steps being demonstrated in the upcoming case study in Section 4, it is worthwhile having a look at the ComGeneration approach [2; 12]. Figure 1 gives an abstract overview.

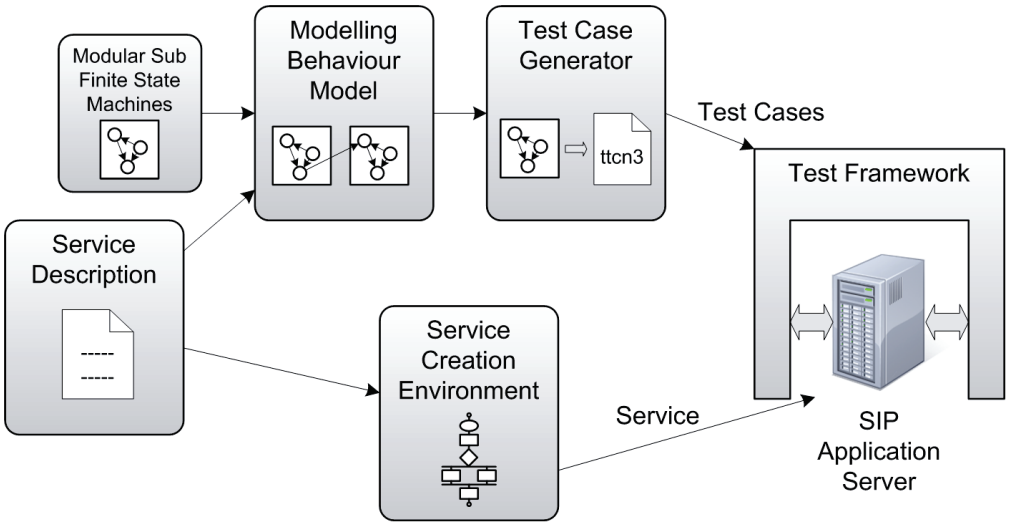

Figure 1. ComGeneration Architecture

The shown architecture can be divided into two main paths: The Service Development and the Test Development. Both the paths have their origin in the initial Service Description that can be seen as a sort of requirements specification for communication services. The Service Description consists of a document containing specific use-case related information and is created by the service provider in consultation with a customer. It contains all possible demands a customer might have for a communication service.

Once the Service Description is defined, both the Service Development and the Test Development can begin in parallel. In the presented approach, it is not ultimately defined which Service Creation Environment (SCE) is used to develop and subsequently deploy a specific service. However, the base for the development and deployment of services with any SCE is the Service Description. The output will then be a service which is deployed on a SIP Application Server.

The Test Development process starts with the test developer who has to interpret the Service Description properly and extracts the relevant service information for the test purpose. Afterwards, he chooses the service-related characteristics out of a repository of so-called predefined modular FSMs. These state machines cover typical service characteristics like protocol sequences for SIP (Session Initiation Protocol) or HTTP (Hyptertext Transfer Protocol). By composing the chosen predefined modular FSMs, the test developer creates a behaviour model, which describes the possible behaviour of a value added service. Once the behaviour model is created it is passed to the Test Case Generator (TCG) which contains an algorithm to automatically generate the service-specific abstract test cases by identifying every possible path through the FSM. After the generation of these abstract test cases is done, they are afterwards converted into executable TTCN-3 (Testing and Test Control Notation) test cases. TTCN-3 is an abstract test scripting language which was standardised by ETSI [13] and ITU-T [14; 15] and supports the modularised creation of test scenarios for message and procedure based systems. In the ComGeneration approach, the execution of the executable TTCN-3 test cases on the deployed service is done within a TTCN-3 test framework.

The significant aspect of this paper is the way how the behaviour model is created by a test developer from the information he can retrieve from the Service Description.

4. CASE STUDY: THE EXAMPLE SERVICE CLICK2IM

In this section an evaluation of the modelling approach is presented with the help of an example service Click2IM (Click-2-Instant-Message).

4.1. Scenario Description

The function of the Click2IM service is to send a SIP MESSAGE with a specified text to a SIP phone having a specified SIP URI. Both the text and the SIP URI are input parameters in a form on a website. Once the parameters are sent by actuating a button, the service creates the SIP MESSAGE which contains the input text and sends it to the SIP phone with the SIP URI.

4.2. Service Description

As described in Section 3, the Service Description is maybe the most important aspect within the process of creating a service on the one hand and testing it against the requirements on the other hand. The Service Description can be seen as a kind of contract between the customer and a service provider.

In the following Table 1, an exemplary Service Description for the Click2IM service is illustrated. It contains a short textual description of the main functionality of the service. Furthermore, the participating roles are defined as well as the preconditions which have to be fulfilled to trigger the service. Also, the possible postconditions are defined within the table which show the possible consequences of the preconditions.

Table 1. Service Description for Click2IM Service

Short Description		
A website should deliver two input masks. The first input mask should contain the address or telephone number (SIP URI) of any participant and the second one should carry any kind of text. A button should be integrated on the website. When submitting it, the text included in the second input mask should be transferred to the address that was filled in the first input mask. If the SIP URI is not reachable or the text could not be transferred an error should occur on the website. If the transfer worked, a success message should occur.		
Roles	Web Browser [b], SIP UAS [s]	Assignment of the roles
Preconditions	Initiator sets any destination address Initiator sets text input Initiator confirms inputs	[b] [b] [b]
Postconditions	1. Receiver is unknown 2. Receiver does not get text message **3. Receiver gets text message**	[b] [b] [b, s]
Prosa	Initiator wants to send a text message to a SIP phone.	

The third postcondition in the table is specially highlighted, because it represents the required functionality of the service which is defined as target. The other postconditions define how the service should behave when certain errors or malfunctions occur. For both the test developer and the service developer, it is necessary to define additional requirements which demonstrate how the postconditions are reached:

a. Standard errors will be signalised, for instance timeouts (leads to → Postcondition 1)

b. Maximal Length of the destination address can be 128 characters (leads to → Postcondition 2)

c. Maximal length of the text input can be 256 characters (leads to → Postcondition 2)

d. Special characters (except of '@') are not allowed in the destination address (leads to → Postcondition 2)

e. Empty text inputs are not allowed (leads to → Postcondition 2)

For the target (Postcondition 3), there are no additional requirements defined as the limitations are already covered.

4.3. Tool Support

Once the Service Description is available the test developer can start to create the behaviour model. For this purpose, the ComGeneration approach provides three editors which have to be used:

1. Connectivity Editor
2. Test Data Editor
3. Behaviour Model Editor

To enable the modelling of the behaviour model, a lot of preliminary work has to be done. Both the Connectivity Editor and the Test Data Editor can contain service-specific conditions which have to be defined before adding further properties to the Behaviour Model. Within the Connectivity Editor, the test developer can define certain parameters for the service like a component, ports and timers. These parameters are derived from typical TTCN-3 test configurations. A component represents one test component within the test. The communication between the component and a system under test is realised through the connection of the local ports, which can be seen as well-defined interfaces [13]. The defined timers can be typical protocol timers or certain global timers.

In the Test Data Editor, the definition of the test data is done with the help of so-called templates. In dependency of the protocol, the test developer can define the inputs of the headers for certain protocol messages. In SIP, this would be the SIP Requests (e.g. INVITE, MESSAGE) and SIP Responses (e.g. 200 OK, 404 Not Found).

Within the Behaviour Model Editor, a test developer can create the referring behaviour model for a specific service by means of a FSM. This FSM describes in what order and under what conditions the test cases are executed. It represents the predicted reactions of the service which were defined as postconditions in the Service Description.

4.4. Creation of the Behaviour Model

Based on the information the test developer retrieves from the Service Description, he first does the test configuration within the Connectivity Editor. Independent of the service he wants to test, he first has to define a so-called component which can be seen as the central element within the Connectivity Editor. Because the two roles Web Browser (HTTP) and SIP UAS (SIP) are mentioned, the test developer exactly knows that he will need the two ports SIP and HTTP to cover the sending and receiving of messages from these protocols. Then, he defines two global timers which may be used to protect the test from infinite waiting for service responses.

Finally, the test developer has to define so-called message variables. Such message variables usually represent protocol messages, for instance SIP or HTTP Requests and Responses which might occur within the test procedure. Because of the message variable's acquisition to protocols, they are usually used in combination with the defined ports. There are other elements like guards and actions which are not yet supported completely by the editors. Nevertheless, the

modelling can be done. Figure 2 shows the configurations for the Click2IM service within the Connectivity Editor.

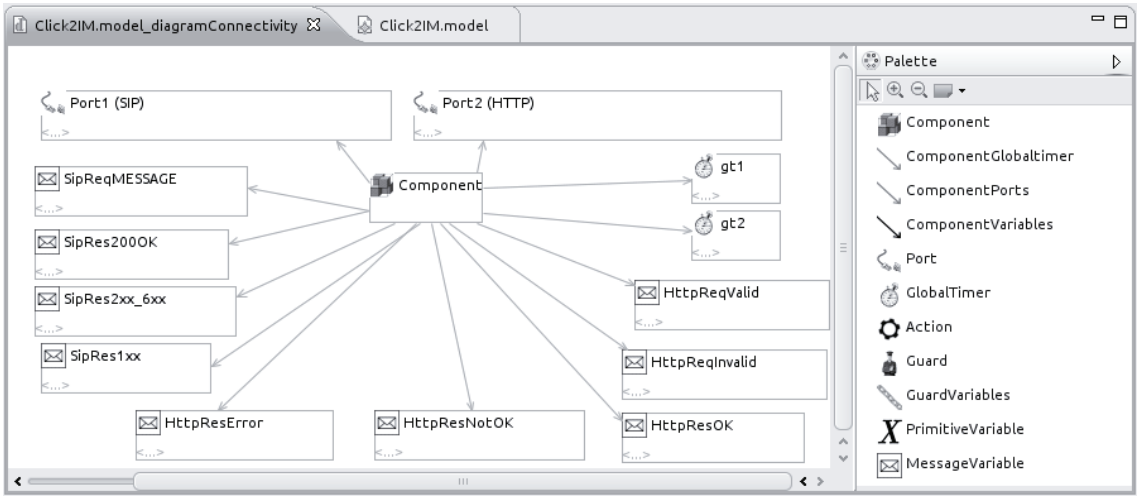

Figure 2. Connectivity Editor for Click2IM

All the Message Variables that were defined in the Connectivity Editor are connected to so-called test data templates. These templates can be edited in the Test Data Editor. For both the protocols SIP and HTTP it is possible to configure a lot of values for headers that were defined in the protocol specifications. Figure 3 demonstrates the configuration of a template belonging to the Message Type SIP Request. Here, the headers for the SIP MESSAGE are edited.

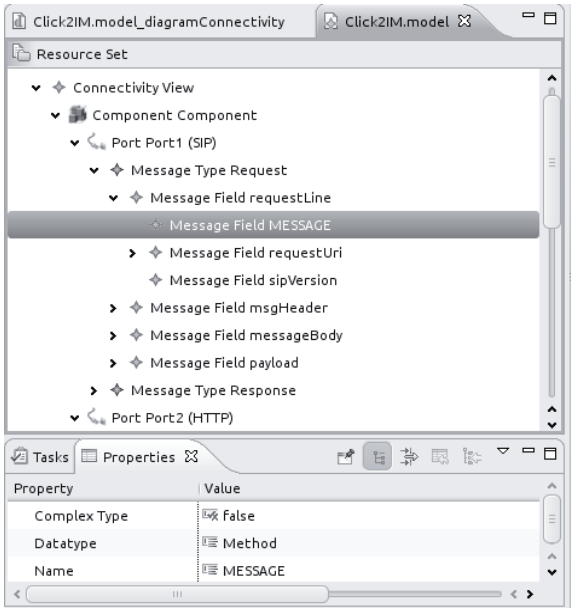

Figure 3. Test Data Editor for Click2IM

Once all the configuration and definition of protocol messages and ports is done, the test developer can start to model the behaviour model. At first he has to choose the relevant sub FSMs for the Click2IM service. Within the Service Description, the roles have been defined. The counterparts of these roles represent the relevant groups of sub FSMs. This would be on the one hand the group of Web Server (HTTP) which consists of two sub FSMs and SIP UAC nonInvite which consists of three sub FSMs.

After choosing the correct sub FSMs, the test developer has to follow the instructions within the Service Description and has to reproduce the behaviour within the composition of the sub FSMs. This composition is done by a concept called Transaction User (TU) which acts as a mediator between possible client and server roles. The whole concept is described in [2].

Altogether, the behaviour model consists of five sub FSMs which have to be composed due to the specification. In the following, the FSM compositions referring to the case "Success" is introduced. From the initial state in the behaviour model, a HTTP POST Message is expected which contains the input text and SIP URI as parameters. Once this is done, the current state is "HttpRequest_Server". Then, the service initiates the sending of the SIP MESSAGE which contains the input text to a SIP phone with the SIP URI. The transition from "HttpRequest_Server" to "SipUAC_nonInvite_init" contains a guard which compares the text inputs of the POST Message and the SIP MESSAGE to verify that they are identical. Afterwards, the transition from "SipUAC_nonInvite_init" to "SipUAC_nonInvite_term" refers to the expecting SIP 200 OK response message which verifies that the SIP MESSAGE has been successfully received by the SIP phone. The last transition is integrated between the states "SipUAC_nonInvite_term" and "HttpResponse_Server". It represents the response of the HTTP Web Server to the originating HTTP POST request of the web browser. Figure 4 demonstrates the complete behaviour model.

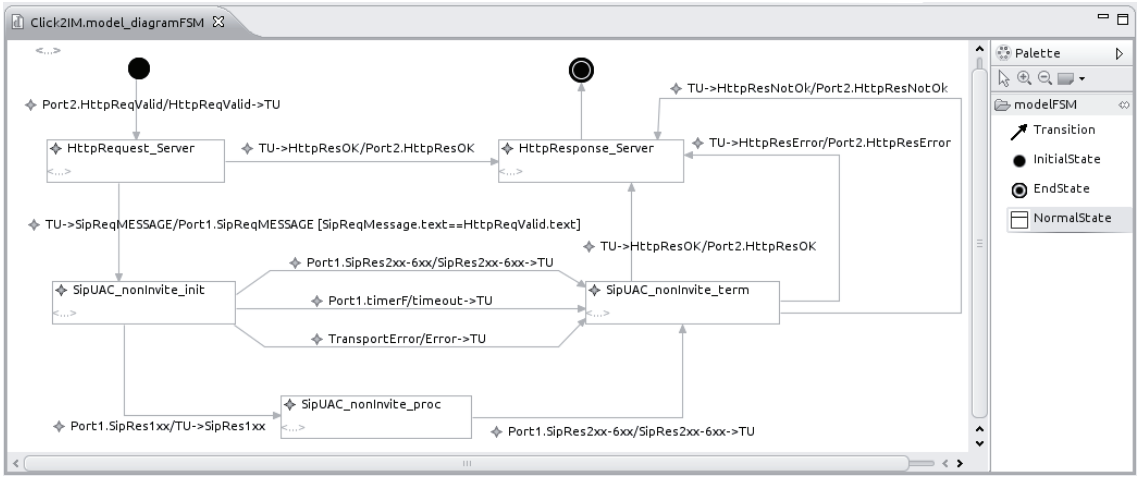

Figure 4. Behaviour Model for Click2IM

5. CONCLUSION

In this paper, we have introduced an approach to automate functional testing of communication services by means of creating a so-called behaviour model from which the relevant test cases are derived. For the modelling purpose, a test developer has to get a deep knowledge about the service requirements from the Service Description and then has to build the behaviour model by composing the sub FSMs. This procedure is simplified and accelerated due to the supply of three editors, the Connectivity Editor, the Test Data Editor and the Behaviour Model Editor. Besides, enhancements like the support of additional protocols and sub FSMs can be easily integrated as the software is modular-based.

Further work should address the improvement of the handling and the support of additional elements within the Connectivity Editor like actions and guards to increase the readability of the Behaviour Model. Moreover, the evaluation of the concept from the Service Description to the execution of test cases on the System under Test, which was not the focus of this paper, is planned near-term.

ACKNOWLEDGEMENTS

The research project ComGeneration providing the basis for this publication was partially funded by the Federal Ministry of Education and Research (BMBF) of the Federal Republic of Germany under grant number 1724B09. The authors of this publication are in charge of its content.

REFERENCES

[1] ComGeneration project website: http://www.ecs.fh-osnabrueck.de/27619.html

[2] Wacht, P.et al. (2011) "A new Approach to design graphically Functional Tests for Communication Services", in *Proceedings of the Fourth IFIP International Conference on New Technologies, Mobility and Security*, Paris, France.

[3] Conformiq website: http://www.conformiq.com/

[4] Yuan, Quilu et al. (2008) "A Model Driven Approach Toward Business Process Test Case Generation", in *Proceedings of the Tenth IEEE International Symposium on Web Site Evolution*, Bejing, China.

[5] Gönczy et al. (2007) "Model-based Testing of Service Infrastructure Components", in *Proceedings of the Nineteenth IFIP TC6/WG6.1 International Conference*, Tallinn, Estonia.

[6] Brucker et al. (2010) "Verified Firewall Policy Transformations for Test Case Generation", in *Proceedings of the International Conference on Software Testing, Verification and Validation (ICST2010)*, Paris, France.

[7] Ali, S.et al., (2010) "Model Transformations as a Strategy to automate Model-Based Testing – A Tool and Industrial Case Studies, Version 1.0", Technical Report.

[8] Wieczorek, S. et al. (2008) "Enhancing Test Driven Development with Model Based Testing and Performance Analysis", in *Proceedings of the Practice and Research Techniques Academic & Industrial Conference TAIC PART*, Cumberland Lodge, Windsor, UK.

[9] Xiaoyan, Zhang, Ming, Huang & Ying, Yu (2008) "OWL-S Based Test Case Generation", in *Journal of Bejing University of Aeronautics and Astronautics*, Bejing, China.

[10] Pretschner, A. et al. (2005) "One Evaluation of Model-Based Testing and its Automation", in *Proceedings of the Twenty Seventh International Conference on Software Engineering*, St. Louis, USA.

[11] Tretmans, G. & Brinksma, H. (2003) "Automated Model-Based Testing", in *Proceedings of the First European Conference on Model-Driven Software Engineering,* Nuremberg, Germany.

[12] Wacht, P. et al. (2010) "Integration of Model-Based Functional Testing Procedures within a Creation Environment for Value Added Services", in *Proceedings of the Sixth Collaborative Research Symposium on Security, E-learning, Internet and Networking (SEIN 2010)*, Plymouth, United Kingdom.

[13] EG 201 873-1 (2008) Methods for Testing and Specification (MTS): The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language . ETSI.

[14] Recommendation Z.140 (2001) The Tree and Tabular Combined Notation version 3 (TTCN-3): Core Language. ITU-T.

[15] Recommendation Z.141 (2001) The Tree and Tabular Combined Notation version 3 (TTCN-3): Tabular Presentation Format. ITU-T.