A New Approach to Design Graphically Functional Tests for Communication Services

Patrick Wacht, Thomas Eichelmann, Armin Lehmann, Ulrich Trick

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

{wacht, eichelmann, lehmann, trick}@e-technik.org

Abstract—**This paper presents a new concept of how a service provider using any kind of Service Creation Environment (SCE) can verify that a generated 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 so-called modular finite state machines by a new composition method. The derivation of the tests is fulfilled by the usage of an adequate path finding algorithm. The following execution of the generated tests is done automatically within a Testing and Test Control Notation (TTCN-3) test framework.**

Key words: Functional Testing; TTCN-3; Behaviour Model; Finite State Machine

I. Introduction

In the near future, network operators and service providers aim for Service Creation Environments (SCE) that enable fast, easy and cost efficient provisioning of value added services. Currently, the building of such SCEs has been done in several research projects, as for instance in the TeamCom project [1; 2]. The TeamCom SCE offers a possibility for developers to design value added services with the help of a graphical user interface and the executable language BPEL (Business Process Execution Language). After the design is fulfilled it is analysed by a code generator and translated into the specific service code. Subsequently, the service can be deployed on an Application Server.

Current SCEs like the TeamCom SCE proved to work properly. However, a very important aspect is usually disregarded by SCEs: the integration of automated functional tests to validate and verify the created value added services.

The enhancement of integrating functional tests will have to be done, because the provider has to assure that the services are executed properly and do not affect other running services within the provider's service environment. Also the integration of testing procedures enables a service provider to check if the built service meets the demands of a customer.

The aim of this paper is to show how testing procedures can be automated. For this purpose, the ComGeneration [3] project has been established that should provide a consistent solution to support the life cycle of a service by simplifying development, testing and provisioning of multimedia communication services. This approach reduces the expenditure of time and cost.

The paper is structured as follows: Section II presents the overall concept of ComGeneration whereas Section III is concerned with the current approach to describe customer's requirements and how to derive the behaviour model from the requirements. Section IV gives an overview of the needed components and mappings to TTCN-3 (Testing and Test Control Notation) [4] for the planned tool chain. Section V discusses the related concepts and works and Section VI concludes the paper.

II. ComGeneration concept

Before looking at the detailed issues about how functional testing of value added services looks like, it is worthwhile having a look at the concept of the ComGeneration approach. Figure 1 gives an abstract overview.

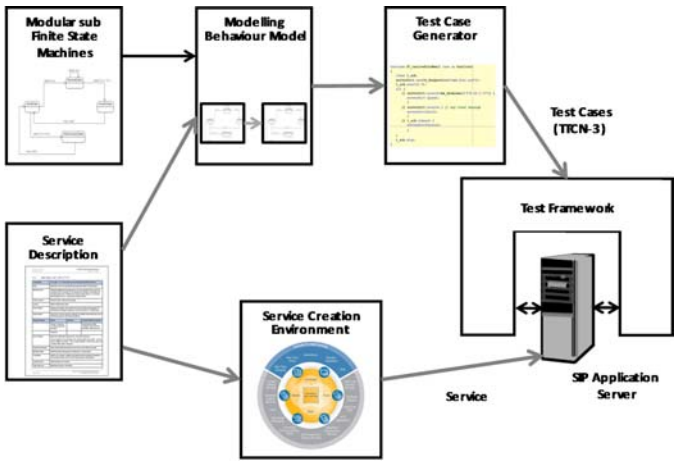

Figure 1. ComGeneration Architecture

The shown architecture can be divided into two main paths: The Service Development and the Test Development. In between there are tasks that are relevant to both paths.

The initial task, which concerns both Service and Test Development, is the definition of a "Service Description". This is a document that can be understood as a requirements specification and is created by the service provider in consultation with a customer. It contains all possible demands a customer might have for a specific value added service.

After the "Service Description" is defined, both the "Service Development" and the "Test Development" are triggered in parallel. The "Service Development" part is fulfilled by a certain SCE. For the ComGeneration project, any SCE can be used to create a service automatically. An exemplary SCE is the TeamCom Service Creation Environment [2; 5]. The service creation within this SCE works as follows: a service designer describes the business

process of the corresponding value added service through a formal control logic based on BPEL. So that the modelling of the business process can be done correctly, it requires the usage of predefined communication building blocks which cover the functionality of typical service aspects. This concept of using elementary communication service components is a key advantage of the approach because it hides the underlying heterogeneous communication networks. Thus, the service designer does not need any detailed knowledge of certain communication protocols and is able to focus on the application logic instead. As BPEL has not been developed for control of real time communication services in heterogeneous networks, a code generator respectively "Service Generator" has been implemented to translate the business process description into Java code. The generated code is based on the Java APIs for Integrated Networks Service Logic Execution Environment (JAIN SLEE) [6] architecture, as this technology fulfils the necessity of communication services. The final step of the approach is the deployment of the code on a specific JAIN SLEE Application Server such as Mobicents [7].

In parallel with the "Service Development" process, the "Test Development" process is initiated by a test developer. First of all, the test developer has to interpret the "Service Description" properly and also has to extract the relevant service information for the test purpose. Afterwards, he has to choose the service related characteristics out of a repository of predefined modular finite state machines. These state machines cover typical service characteristics like protocol sequences for TCP (Transmission Control Protocol), SIP (Session Initiation Protocol) or HTTP (Hypertext Transfer Protocol). By composing the chosen predefined modular finite state machines, the test developer creates a so-called behaviour model, which describes the possible behaviour of a value added service. Depending on the service's complexity, the behaviour model itself is also a more or less complex finite state machine. If the behaviour model is complete, an algorithm generates the service specific test cases by identifying every possible path through the finite state machine. After the generation is done, every identified test case is converted to TTCN-3 [4] within the "Test Case Generation" process. TTCN-3 is an abstract test scripting language which was standardized by ETSI [8; 9; 10] and ITU-T [11; 12] and supports the modularized creation of test scenarios for message and procedure based systems. In the ComGeneration approach, the execution of the generated TTCN-3 test cases on the deployed service is done within a TTCN-3 test framework.

One very significant aspect of this paper is on the one hand side the way how the behaviour model is modelled by a test developer from the content of the "Service Description". On the other hand side, the automatic generation of test cases from the behaviour model and the following execution on the System under Test (SUT) is the main focus. The mentioned aspects should verify that a communication service meets the requirements of a customer. The following Figure 2 shows the relevant steps for the "Test Development" in detail.

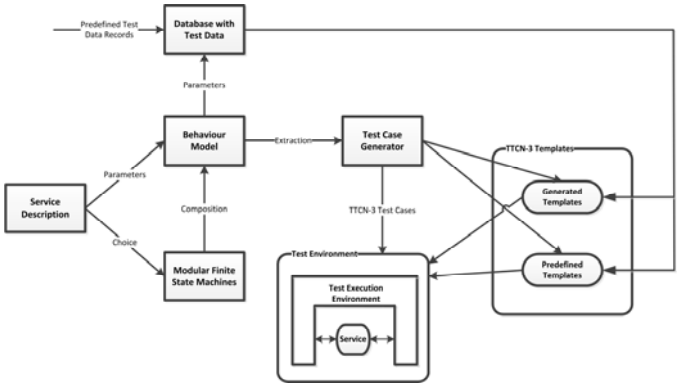

Figure 2. Test Development Process

On the basis of the description, the behaviour model is described by a finite state machine. Some important service related parameters can be specified within the "Service Description". These parameters have to be integrated into the behaviour model. In TTCN-3, parameters and their values are defined as so-called TTCN-3 templates. This leads to the fact that every parameter within the behaviour model has to be transformed to a TTCN-3 template. An example for a relevant parameter within the behaviour model could be the name of a SIP instant message (e.g. "MESSAGE"). This could mean that during the service flow such a message is expected to be sent, maybe to a specific SIP User Agent. The information of possible message structures used within the behaviour model has to be available during the "Test Development" process. So, a database with test data is required. In this database, many possible test data records are predefined as TTCN-3 templates. Predefined templates already exist in the database, even before the behaviour model was created. Depending on which finite modular state machines are used within the model, the predefined templates are activated and integrated within the test framework. The generated templates are completely new. They are associated to the parameter inputs made by the test developer. The last step of the "Test Development" process is the testing of the service itself within the test framework.

III. Service Description and Behaviour Model

As depicted in the last section, the test developer has to design a behaviour model based on the information he could retrieve from the service description. Both the service description and the model will be explained in the following.

A. Service Description

The aim of the service description is to deliver a complete set of requirements from the view of a user which has to be fulfilled by the communication service. A user can on the one hand be a person or on the other hand be an external system.

For the ComGeneration project, a specific way of defining a service description has been developed. It has been derived from a standardised object oriented method and includes the following steps:

1. Short description
2. Identification of the roles
3. Requirements specification

4. Enhanced requirements specification
5. Identification of the communication interfaces

The initial task is about writing a very short description of the service's functionality which mentions the elementary usage of the service.

The short description is followed by the second step, the identification of the roles or rather participants. Such roles, where users are able to communicate with the service, can be defined as views on the service. Regarding often used protocols in communication services like HTTP or SIP, roles which refer to these protocols could be a web browser for HTTP and a softphone for SIP.

After the roles have been defined, the requirements specification has to be established. Such a specification contains significant cases that may occur when using the communication service. Usually, the definition of these cases requires the cooperation of the customer and the service provider. A description of a case usually contains the following components: user role, preconditions, target, post conditions.

The relevant user role for the case is selected from the identified service roles. The preconditions describe the situation (e.g. 'SIP URI entered'), which leads to the designated upcoming situation. The target always relates to the status of the role which has to show a reaction (e.g. 'Softphone reachable'). Finally, the post conditions should forecast the possible situations which can occur.

Once the requirements specification has been finalized, some enhanced requirements are defined without the customer in step four. Here, some specific information can be defined such as the maximal length of SIP URI etc.

In the last step, the communication interfaces for the service are defined which relate to the specified roles.

As the service description is created by human hand, it might be ambiguous and error-prone. To reduce occurring problems, the whole creation process is simplified and standardized by providing the service provider and the customer with a so-called "Requirements Catalogue". Such a catalogue contains predefined standards, restrictions, requirements and possible roles. The selection of these predefined aspects within the described five steps results in a service description.

B. *Behaviour Model*

In order to do functional testing of a communication service, a test developer has to know, how the service should behave according to the specification. In the ComGeneration approach, this knowledge can be retrieved from the service description to build a behaviour model.

The short description within the service description delivers the test developer the abstract overview of the service. The identification of the roles and communication interfaces enables him to choose the service-relevant modular finite state machines from a repository. Eventually, the requirements specifications and the enhanced requirements specification represent the service behaviour. From these specifications, the test developer learns how to compose the chosen modular finite state machines to a more complex finite state machine, the behaviour model. The composition of two or more modular finite state machines is realised by a new concept called the Transaction User (TU). The TU always acts as a mediator between possible client and server roles. Hence, every predefined modular finite state machine has interfaces where the TU acts as a sender or receiver. The TU does not contain any information about the implementation of the service, but it allows the description of a service from the view of the SUT. The following Figure 3 gives an example of a composition of a two modular finite state machines, HTTP_Server and SIP_UAC_nInvite.

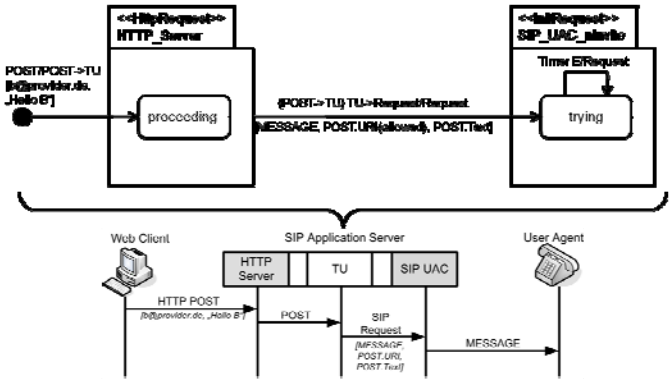

Figure 3. Composition of modular finite state machines with TU

The composition describes a HTTP POST Request, which contains a SIP URI ('b@provider.de') and a text ('Hello B'). When the POST Request is received by the HTTP Server of the SIP Application Server, the TU is informed and the TU instructs the SIP UAC (User Agent Client) with respect to the service logic to send a SIP Request or rather a SIP MESSAGE with the contents of the POST Request.

IV. Components, Mapping and Toolchain

Before the mapping to TTCN-3 and the tool chain will be explained, first the components, the so-called modular finite state machines, are described.

A. *Modular finite state machines*

The modular finite state machines, which establish the basis for the behaviour model, are predefined and reusable components which are usually based on specific protocols (SIP, TCP, HTTP) or categories (databases). The structures of the state machines for the protocol are derived from the particular protocol specification. The following Figure 4 shows an exemplary state machine.

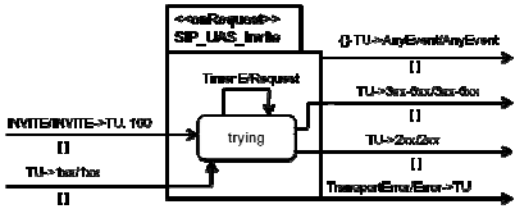

Figure 4. Structure of the finite state machine SIP_UAS_Invite

Depending on the specification, each state machine can have several inputs and outputs. These interfaces are used to compose more state machines with each other and to enable the building of the behaviour model.

The state machine SIP UAS_INVITE which was derived from [13] describes the handling of an incoming SIP INVITE message for an User Agent Server (UAS). Every incoming and outgoing transition represents a message which either is received by the UAS or sent. The possible responses, which can be initiated by the User Agent Server, are defined as outputs. Besides the relevant protocol specific outputs like the SIP status codes (2xx, 3xx-6xx) and occurring transport errors, there is also a so-called "AnyEvent" defined. This output can be understood as a placeholder for any kind of message from any protocol. This technique enables the composing of all available state machines.

A test developer only knows about the available state machines from specific protocols. His main task to build a behaviour model is to handle the interfaces of the finite state machines.

B. Mapping to TTCN-3

The test machine generates messages and sends them to the system under test (SUT) or it checks the messages sent by SUT and responds to those messages. With the messages sent to the SUT, the test system tries to provoke errors on the SUT. This requires the test machine to understand the messages from the SUT and also provide test data that consist of positive and negative cases. To support these tasks TTCN-3 defines some elements that are required to create tests. In Table I, some of these elements are described.

The Test Case Generator (TCG) creates test cases from the FSMs and translates them into TTCN-3 code. So the TCG needs some knowledge about the TTCN-3 elements or generates these elements by himself. For the mapping of the elements from the TCG to TTCN-3 two concepts are introduced, the Connectivity Concept and the FSM Concept. To generate the required test cases the TCG needs to obtain the information from both concepts. Each concept provides other information for the TCG and the two concepts together offer all the required information to generate the TTCN-3 code.

TABLE I. TTCN-3 ELEMENTS

TTCN-3 element	Description
Type Definition	Defined data types to describe the exchange of data between test components and SUT.
Port Definition	Communication between the test components and the SUT is established by connecting local ports. e.g. SIP port, HTTP Port
Component Definition	Structur of the test components that represent the client and server protocol-specific endpoints e.g. UAS SIP, HTTP Client
Test Case	Runs individual test components Summarizes the test events
TTCN-3-Templates	Definition for the description of test data
Control Part	Describes the order and the conditions for the execution of individual test cases
TTCN-3-Codecs and Test Adapter	Adaptation for the SUT Converts TTCN-3 code in an understandable format for the SUT
Verdict	Judgement (Pass, Fail, Inconclusive, error)

The FSM Concept is presented in Figure 5.

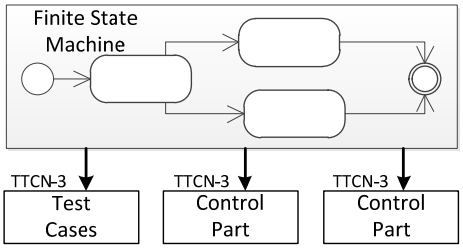

Figure 5. Finite State Machine Concept

From the FSM, different TTCN-3 test cases can be generated. The FSM also describes in what order and under what conditions the test cases are executed. From this information, the TTCN-3 Control Part is generated. The FSM represents only the positive "good" reactions of the service. All other cases, which occur in TTCN-3, are defined as invalid (failed). The TTCN-3 verdict can therefore also be derived from the FSM approach.

The Connectivity Concept is shown in Figure 6.

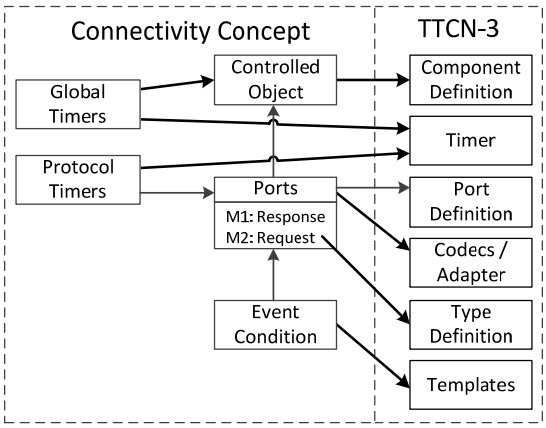

Figure 6. Connectivity Concept

Also from this concept, information for the mapping to TTCN-3 elements can be derived. The concepts are connected through the Controlled Object (CO). A CO is assigned to the FSM within the FSM concept. This CO holds information about the used ports and timers.

The TCG supports different timers (protocol timer and global timer). The protocol timers are pre-defined and belong to the respective ports. Depending on the protocols or the messages which are required by the test developer, the associated protocol timers are automatically added to the tests. The test developer also has the ability to define its own timers. These timers are called global timers, because they are defined and can be manipulated within the whole FSM.

The ports in the TCG map to the TTCN-3 ports. For all supported protocols, the ports are predefined in the TCG. Also, all possible protocol messages must be predefined within the respective ports. The timers which are defined within the protocol specifications are also added to the port definition. The definition of the test data in TTCN-3 is done with the help of templates. These TTCN-3 templates are generated by the TCG. For this purpose the TCG uses so called Event Conditions. The test developer uses these Event Conditions to choose from pre-defined test data or to define its own test data.

The data types which are used are predefined and correspond to the TTCN-3 Type Definitions. The ports used in the TCG are assigned to the corresponding TTCN-3 codec and TTCN-3 adapter.

C. *Toolchain*

The TCG generates the test cases and test data for TTCN-3 from the FSM which is created by the test developer. The TCG is composed by several elements, the GUI, the FSM Parser, the Connectivity Parser, the FSM Pathfinder and the TTCN-3 Code Generator, see Figure 7. In order to model the FSM in a simple manner, the test developer is supported by a GUI. This GUI consists of several views, the FSM View and the Connectivity View. In the FSM View, graphical FSM states and transitions can be defined.

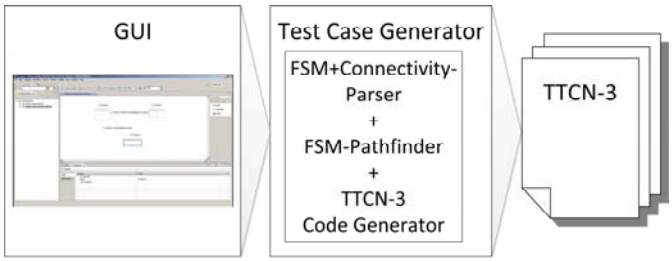

Figure 7. Tool chain

In the Connectivity View, ports, messages and timers are added to the Controlled Object and test data is defined or adjusted. Another element of the TCG is the parser. The parser analyses the FSM and gathers the information needed for the TTCN-3 code generation. All relevant test cases are required for the SUT test. A test case represents a path from initial state to end state within the FSM. This means that all paths within the FSM are discovered by the pathfinder. From each resulting path of the FSM a TTCN-3 test case is generated. With all of the test cases and the information obtained by the parser, a TTCN-3 code generator generates the complete TTCN-3 test.

V. Related Work

The approach to describe tests with finite state machines is quite common. Conformiq [14] 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. Furthermore, there are no predefined state machines which can be used to simplify and accelerate the modelling process.

A similar approach from Yuan [15] describes test case generation from UML activity diagrams, but the main focus is about testing Web Service compositions with the help of TTCN-3. However, the functionality is quite limited, as only HTTP is supported which enables Web Service composition.

VI. Conclusion

In this paper, we have introduced a new approach to integrate automated functional testing within any SCE to validate that a created value added service meets the requirements of the customer who ordered the service. Therefore, we developed a technique to create a behaviour model for a service from predefined modular finite state machines with the help of a certain concept, the Transaction User. From the behaviour model, abstract test cases can be derived by the usage of an efficient path finding algorithm. The abstract test cases are then converted to executable test cases in TTCN-3, a standardised scripting language used in testing of communication services.

Further work should address the improvement of the path finding algorithm and validation of the concept.

Acknowledgment

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 grand number 1724B09. The authors of this publication are in charge of its content.

References

[1] TeamCom project website: http://www.ecs.fh-osnabrueck.de/teamcom.html
[2] A. Lehmann et al.: “TeamCom: A Service Creation Platform for Next Generation Networks”, IEEE ICIW 2009, Venice/Mestre, Italy, 24-28 May 2009.
[3] ComGeneration project website: http://www.ecs.fh-osnabrueck.de/27619.html
[4] ETSI website for the TTCN-3 standards: http://www.etsi.org/WebSite/technologies/ttcn3.aspx
[5] T. Eichelmann et al.: ”Creation of value added services in NGN with BPEL”, Proceedings of the Fourth Collaborative Research Symposium on Security, E-learning, Internet and Networking (SEIN 2008), Wrexham, United Kingdom, 5-9 November 2008.
[6] Sun Microsystems, Open Cloud, JSR-000240 Specification, Final Release, "JAIN SLEE (JSLEE) 1.1", SUN, 2008.
[7] Mobicents Open Source JAIN SLEE Server, http://www.mobicents.org
[8] EG 201 873-1: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part1: TTCN-3 Core Language. ETSI, September 2008.
[9] EG 201 873-2: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part2: TTCN-3 Tabular presentation Format (TFT). ETSI, February 2007.
[10] EG 201 873-3: Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part3: TTCN-3 Graphical presentation Format (GFT). ETSI, February 2007.
[11] Recommendation Z.140: The Tree and Tabular Combined Notation version3 (TTCN-3): Core Language. ITU-T, July 2001.
[12] Recommendation Z.141: The Tree and Tabular Combined Notation version3 (TTCN-3): Tabular Presentation Format. ITU-T, July 2001.
[13] J. Rosenberg et al. : RFC 3261 – SIP : Session Initiation Protocol. IETF, June 2002.
[14] Conformiq website: http://www.conformiq.com/
[15] Quilu Yuan et al. : A Model Driven Approach Toward Business Process Test Case Generation. IEEE, June 2008