Integration of Model-Based Functional Testing Procedures within a Creation Environment for Value Added Services

P.Wacht[1,2], A.Lehmann[1,2], T.Eichelmann[1,2], W.Fuhrmann[3], U.Trick[1] and B.V.Ghita[2]

[1]Research Group for Telecommunication Networks, University of Applied Sciences Frankfurt/M., Frankfurt/M., Germany
[2]Centre for Security, Communications and Network Research, University of Plymouth, Plymouth, United Kingdom
[3]University of Applied Sciences Darmstadt, Darmstadt, Germany
e-mail: wacht@e-technik.org

Abstract

Actual Service Creation Environments (SCE) do not support functional testing of automatically created value added services. This leads to a problem as there is no verification that the service is created properly according to the requirement's specification. This paper presents an approach to integrate a testing framework into an existing SCE, which enables systematic and effective functional testing of value added services. The procedure is based on the idea that a behaviour model is created from which the amount of test cases for a specific service can be derived. The identified test cases are transferred to TTCN-3 (Testing and Test Control Notation 3) code and executed on the created service, which is the SUT (System under Test).

Keywords

SCE (Service Creation Environment); finite state machines; functional test automation; TTCN-3 (Testing and Test Control Notation 3)

1. 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 in the TeamCom project (TeamCom, 2009; Lehmann et al., 2009). 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.

The TeamCom approach proved to work properly for several services. However, a very important aspect is not yet supported by the SCE: the integration of automated functional tests to validate and verify the created value added services. This enhancement of the SCE will have to be done, because functional tests are derived

from the service's specification, which contains the customer's requirements and wishes for the service. So 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 integrated within SCEs systematically. For this purpose, the ComGeneration (ComGeneration, 2010) 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.

A similar approach to ComGeneration was accomplished in the project TT-Medal (TT-Medal, 2010), which has proven the advantages of using UML to generate TTCN-3 tests. Also, the firm Conformiq (Conformiq, 2010) implemented a test suite (Conformiq Tool Suite) which automates the design of functional tests for software and systems. However, the handling of the suite requires deep knowledge in UML and in several programming languages.

The content of this paper is structured as follows: In section 2 the consistent concept of the service and test platform is described. The 3rd section is concerned with the actual approach to describe customer's requirements within a "Service Description". Section 4 describes the relevant parts of the "Test Development" process and in section 5 the execution of test cases is introduced. Finally, section 6 offers a conclusion.

2. Service Creation and Testing Environment

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.

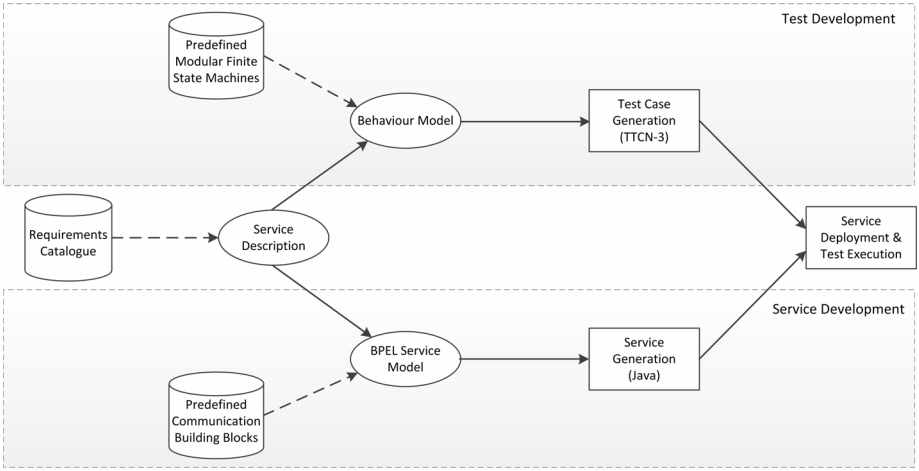

Figure 1: Service and test development for value added services

The shown architecture can be divided into two main layers: The Service Development and the Test Development. In between there are tasks that are relevant to both layers. The kinds of shapes illustrated in the picture have a special meaning, for instance, the container-like shapes generally represent predefined data which a person being involved in the process can choose from. This is shown by the dashed arrow in Figure 1. The circle shapes define actions where a human has to be involved. In contrast, the rectangle shapes only represent tasks that are fulfilled automatically without human interaction.

The initial task that 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. To simplify the creation process of the "Service Description", the service provider provides the customer with a so-called "Requirements Catalogue". This catalogue contains predefined standards, restrictions and requirements. The selection of these predefined aspects for a specific value added service results in a form of service description. Furthermore, the relevant roles for the usage of a service are identified within this document.

After the "Service Description" is defined, both the "Service Development" and the "Test Development" are triggered in parallel. The "Service Development" part already exists in the TeamCom Service Creation Environment (Eichelmann et al., 2008). The service creation within TeamCom 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 JAIN SLEE (Sun and Open Cloud, 2008) 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 (Mobicents, 2010).

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 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. A behaviour model can be seen as complete, if all the requirements specified in the "Service Description" are covered within the model.

After the generation is done, every identified test case is converted to TTCN-3 (Testing and Test Control Notation Version 3) (TTCN-3, 2010) within the "Test Case Generation" process. TTCN-3 is an abstract test scripting language which was standardized by ETSI (ETSI, 2010) and ITU-T (ITU-T, 2010) 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.

3. Service Description

Defining the "Service Description" for a specific value added service is maybe the most important aspect within the process of creating a service, because it can be seen as a kind of contract between a customer and a service provider. It enables the customer to communicate his requirements for a service to the service developer so that the service can be realized properly. For the ComGeneration project, a specific way of defining a "Service Description" has been developed. It has been derived from a standardized object oriented method and includes the following steps:

1. Short description
2. Identification of the roles (without the system)
3. Requirements specification (with customer)
4. Enhanced requirements specification (without customer)
5. Identification of the communication interfaces

The initial step is to write a very short description about the service's functionality. Exemplarily, this is shown for the Web2IM (Web-to-Instant-Message) service:

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 web site. 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 couldn't be transferred an error should occur on the web site. If the transfer worked, a success message should occur.

This short description of the service is followed by the second step, the identification of the roles respectively participants. For the Web2IM example, this would be on the

one hand a web browser ([B]) and on the other hand a text display unit. As SIP is used to transfer the text, the display unit could be a SIP softphone ([S]).

The third step to define a "Service Description" requires the cooperation of the customer and the service developer. Both define significant cases that may occur when using the service. The table illustrates a possible case for the Web2IM service.

		Role
Preconditions	Website available	[B]
	SIP URI entered	[B]
	Text entered	[B]
	Entry approved	[B]
Target	Softphone reachable	[S]
Postcondition	Softphone gets text	[S]
	Approval is displayed	[B]
Description	After accessing the website, SIP URI and text are entered. Entries are approved and text is delivered to the softphone with the SIP URI. The receipt is approved on the website.	

Table 1: Standard case for requirements specification of Web2IM service

Depending on the kind of service, a few of such cases may have to be identified. Afterwards, some enhanced requirements are defined without the customer in step 4 of the "Service Description" process. Here, some specific information is defined such as the maximal length of the SIP URI or the input text.

In the last step, the communication interfaces for the service are identified. This is very helpful information for the test developer, because he will then be able to choose the relevant modular finite state machines from the repository to build a behaviour model. For the Web2IM service, the communication interfaces are the following two: HTTP Client and SIP UAC nonInvite.

4. Test Development

The most significant aspect of this paper is the generation and execution process of test cases for specific value added services to verify that they meet the demands of the customer's requirements. For this purpose, Figure 2 shows the relevant steps for "Test Development" in detail.

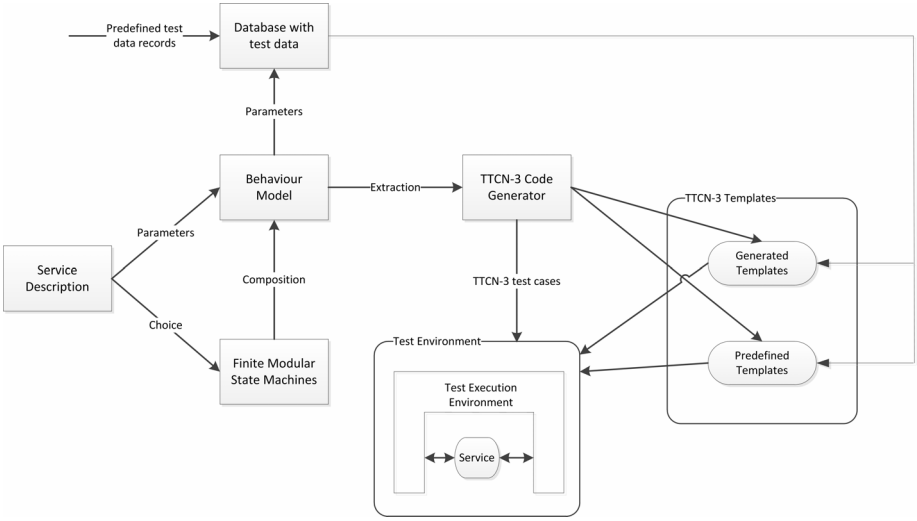

Figure 2: Test Development process

As already shown in Figure 1 and now also in Figure 2, the first condition to start the process is an existing "Service Description". On the basis of the description, the relevant finite modular state machines are chosen and composed to a behaviour model. It was already mentioned in the previous section that some important service related parameters can be specified within the "Service Description" that also have to be integrated into the behaviour model. In TTCN-3, parameters and their values are defined as 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, e.g. 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. These templates can be enhanced by the data from the behaviour model. One has to distinguish between predefined and generated 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. This can only be done if all the extracted test cases exist as TTCN-3 test cases and all the relevant TTCN-3 templates were activated respectively generated and integrated into the test environment.

4.1. Modular finite state machines

Before the structure of the behaviour model is introduced, first the components, the modular finite state machines, are described. The finite state machines are predefined and reusable components which are usually based on specific protocols (SIP, TCP, HTTP) or categories (databases). The structure of the finite state machines for the protocols is derived from the particular protocol specification. Depending on the specification, each finite state machine can have several inputs and outputs. These interfaces are used to compose more finite state machines with each other and to enable the building of the behaviour model.

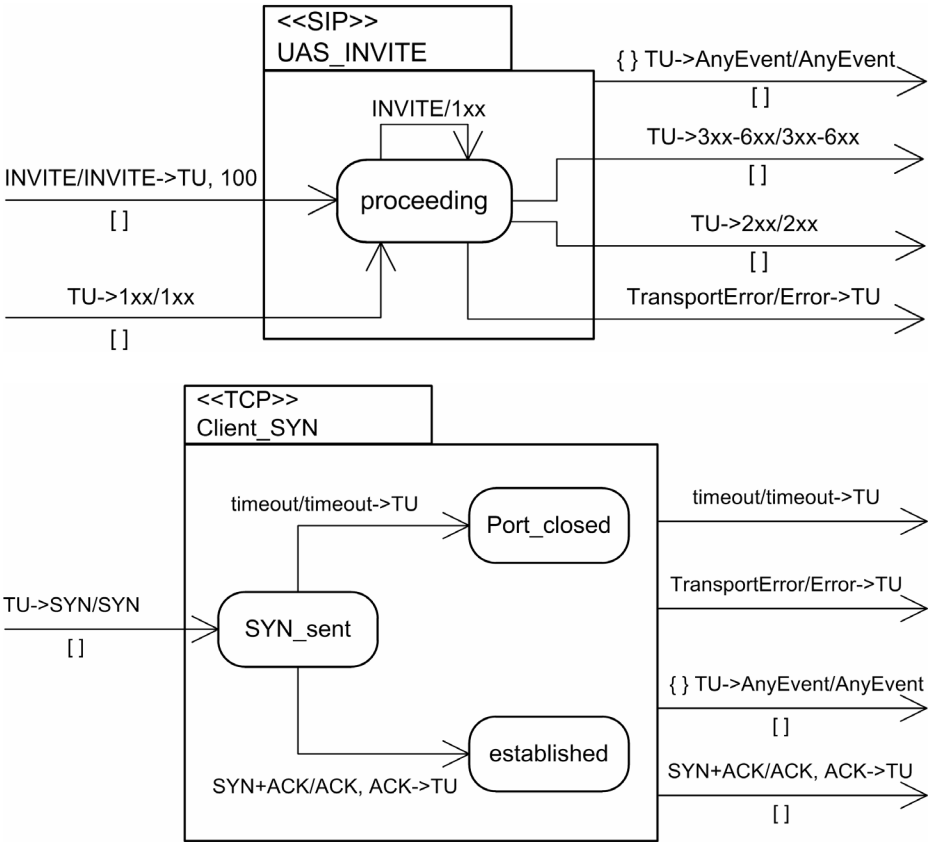

Figure 3: Structure of the finite state machines SIP UAS_Invite and TCP Client_SYN

Figure 3 shows exemplarily the structure of the finite state machines with the help of the two components SIP UAS_INVITE and TCP Client_SYN.

The finite state machine SIP UAS_INVITE describes the handling of an incoming SIP INVITE message for a User Agent Server (UAS). Every incoming and outgoing

transition represents a message that either is received or sent by the UAS. 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 finite state machines.

The structure of the finite state machine TCP Client_SYN is a little bit different from the SIP UAS_INVITE, as there are three existent states within the finite state machine. The internal transitions between these states are fixed and always used in the same manner. The TCP Client_SYN represents a TCP connection establishment. The meaning of the “TU” statement within the transitions is discussed in the following section 4.2.

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

4.2. Behaviour model

In order to do functional testing of a value added service, a test developer has to know, how the service should behave according to the specification, if, for instance, certain messages occur. This knowledge can be retrieved from the “Service Description”. If the understanding of the service is fulfilled, the test developer chooses the relevant modular finite state machines and composes them to get the behaviour model. The composition of finite state machines is the only changing component, the internal transitions, however, are unchangeable. In order to assure, that a behaviour model can be established, a new concept, the Transaction User (TU), was installed. The TU perceives itself as a switching unit between the possible roles of an Application Server (AS) as User Agent Server and User Agent Client. Concurrently, the TU is a connector between modular finite state machines. It enables the test developer to reproduce the service logic for test purposes.

An example of composing the finite state machines from different protocols (Figure 3) is shown in Figure 4. This service logic could be interpreted as follows: the service expects calls from a selection of User Agents (alice, bob). Only if these User Agents call the service, a TCP connection to a specific socket (IP, Port), e.g. to an external database, is established. Here, the SIP INVITE is a sort of trigger.

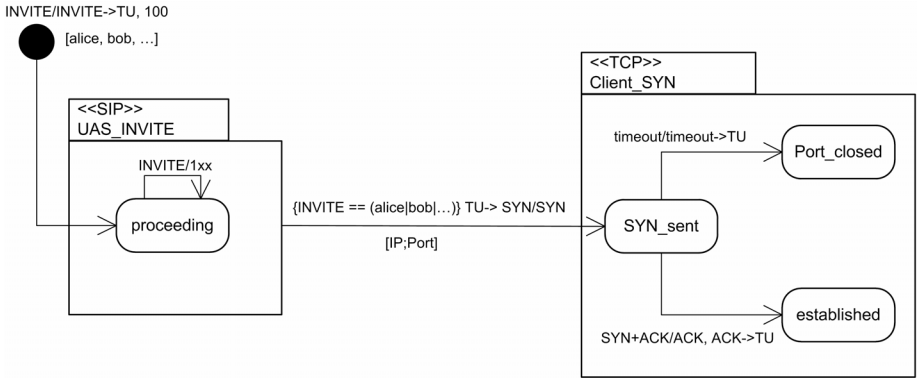

Figure 4: Exemplary role of the TU as a connector of finite state machines

The composition of the two finite state machines is done by using the input message from the TCP Client_SYN as the AnyEvent output message of the SIP UAS_INVITE. Figure 5 clarifies the whole concept of the TU.

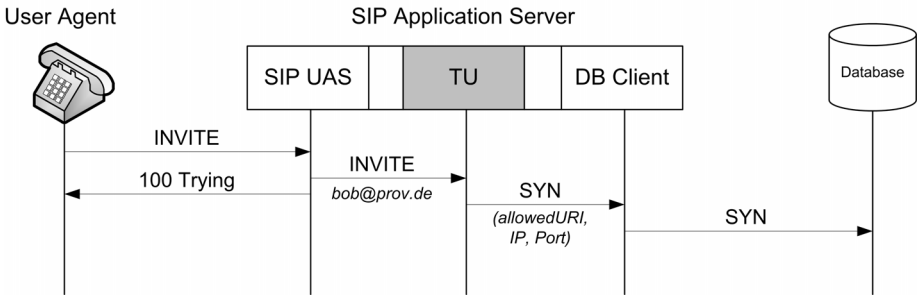

Figure 5: Equivalent message flow by traversing the behaviour model

The demonstrated message flow in Figure 5 reflects the transition path within the finite state machine shown in Figure 4.

When the test developer creates a behaviour model for a value added service he does not need to have any information about the insides of a finite state machine, because he only has to handle the interfaces and has to set relevant parameters. Figure 6 demonstrates a simplified but complete behaviour model of the Web2IM service which was introduced in section 3. The two HTTP modular finite state machines, Server_Req and Server_Resp, represent the initiation of the POST request and the expected responses from the server. In contrast, the three SIP modular finite state machines describe the behaviour of the SIP Message being sent by the service. As the service is the sender of the SIP Message, only the UAC finite state machines are considered.

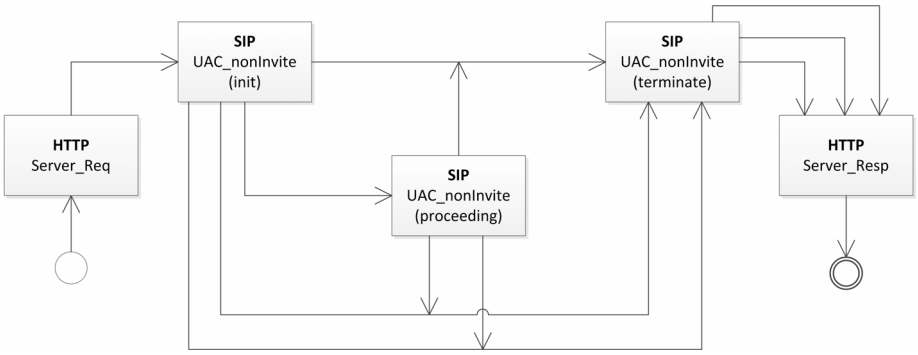

Figure 6: Behaviour model for the value added service Web2IM

4.3. Test Case Generation

The main argument for using finite state machines as behaviour models is that the transition paths within a state machine represent test cases for a value added service. Therefore, an algorithm has to be defined that identifies all the possible paths. Such an algorithm has not yet been realized as the implementation phase of the ComGeneration project has started recently.

The path finding within the behaviour model can be associated to the following criteria:

- state coverage: every state has to be visited once
- transition coverage: every transition has to be passed once
- event coverage: every possible event has to occur once

For every above-mentioned criterion, the identification of paths respectively the generation of test cases is fulfilled. Afterwards, when the test cases are available, they are transferred to real TTCN-3 test cases by a TTCN-3 code generator.

5. Test Execution

After the generation of the TTCN-3 test cases for a specific value added service has been done, the test cases have to be executed on the service respectively the System under Test (SUT). For this purpose, a TTCN-3 test execution environment is required. Within the ComGeneration project, the integrated test development and execution environment TTworkbench is used, which was developed by Testing Technologies (Testing Technologies, 2010). In order to connect the test execution environment to the SUT, a system adapter is required. Such a system adapter contains adapters that are relevant to enable the communication with the SUT.

Using the example of the Web2IM service, which has been introduced in the previous section, the system adapter would possibly contain a UDP adapter and a HTTP adapter. The UDP adapter is responsible for transferring SIP messages and it is configured to map the TTCN-3 ports to the UDP ports. As in TTCN-3 messages are defined as data structures, the test case executives will use the SIP codec for encoding the data structures to real SIP text messages and vice versa. For the usage of HTTP requests and responses that are used to trigger the service, there is also an adapter required.

Before the test cases can be executed, PTCs (Parallel Test Component) have to be configured that represent the relevant endpoints. For the Web2IM service, two different PTCs have to be defined. The first PTC sends the initial HTTP request to the SUT which contains the text message and the SIP URI and receives the HTTP response with the failure or success message. In contrast, the second PTC receives the SIP MESSAGE from the SUT and answers with a specific SIP status code.

Figure 7 illustrates the structure of the TTCN-3 testing environment for the Web2IM service.

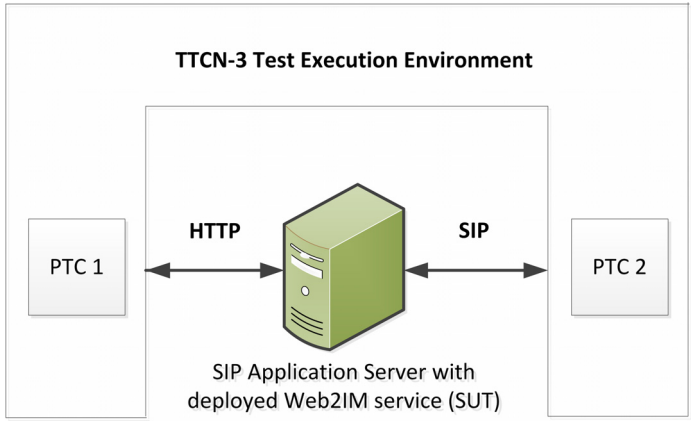

Figure 7: Test Execution Environment for Web2IM service

6. Conclusion

In this paper, we have introduced an approach to integrate functional testing within an existing SCE to validate that a created value added service meets the requirements of the customer who ordered the service. For the testing purpose, a test developer has to get a deep knowledge about the service requirements from the "Service Description" and then has to build an abstract model in the form of a finite state machine, the behaviour model. Although the creation of such a behaviour model tends to be complicated, the advantages dominate, because the extraction of test cases from the model can be done easily with an adequate algorithm.

Once the behaviour model has been developed, it takes very little time until the execution of the generated test cases on the SUT can be done. With the support of quite a lot of communication protocols like SIP, TCP or HTTP, many sorts of value added services can be tested. Besides these positive effects of such an implementation, the test developer who uses the tool has to have a deep knowledge of every used protocol.

As soon as the ComGeneration development environment is implemented, which enables the creation of a behaviour model, an evaluation of the approach is required. To prove the reduction of time and costs in comparison with manual testing, both procedures have to be accomplished for a couple of exemplary value added services.

7. 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.

8. References

ComGeneration Project Web Site (2010): http://www.ecs.fh-osnabrueck.de/27619.html. (Accessed 15 August 2010)

Conformiq (2010): http://www.conformiq.com . (Accessed 20 August 2010)

Eichelmann, T., Fuhrmann, W., Trick, U. and Ghita, B. (2008), “Creation of value added services in NGN with BPEL”, *SEIN*, Wrexham, 2008

ETSI Testing and Test Control Notation (TTCN-3) (2010): http://www.etsi.org/WebSite/technologies/ttcn3.aspx. (Accessed 14 August 2010)

ITU-T – The Evolution of TTCN (2010): http://www.itu.int/ITU-T/studygroups/com07/ttcn.html (Accessed 14 August 2010)

Lehmann, A., Eichelmann, T., Trick, U., Lasch, R., Ricks, B. and Tönjes, R. (2009), “TeamCom: A Service Creation Environment for Next Generation Networks“, ICIW, Venice, 2009

Mobicents Open Source JAIN SLEE Server Project Web Site (2010): http://www.mobicents.org. (Accessed 14 August 2010)

Sun Microsystems, Open Cloud (2008), JSR-000240 Specification, Final Release, “JAIN SLEE (JSLEE) 1.1”, Sun.

TeamCom Project Web Site (2009): http://www.ecs.fh-osnabrueck.de/teamcom.html. (Accessed 14 August 2010)

TTMedal (2010): http://www.tt-medal.org, (Accessed 20 August 2010)

Testing Technologies (2010): http://www.testingtech.com. (Accessed 15 August 2010)

TTCN-3 application areas (2010): http://www.ttcn-3.org/ApplicationAreas.htm. (Accessed 14 August 2010)