A New Service Description for Communication Services as Basis for Automated Functional Testing

P. Wacht, U. Trick
Research Group for Telecommunication Networks
University of Applied Sciences Frankfurt
Frankfurt, Germany
{wacht, trick}@e-technik.org

W. Fuhrmann
University of Applied Sciences Darmstadt
Darmstadt, Germany
w.fuhrmann@fbi.h-da.de

P. Wacht, B. Ghita
Centre for Security, Communications, and Network research
University of Plymouth
Plymouth, United Kingdom
{patrick.wacht, bogdan.ghita}@plymouth.ac.uk

***Abstract*—The advances in the telecommunication domain to support complex communication services has resulted in a need for a new approach to automatically verify that the communication services meet the demands of the customers. This paper presents a concept for automated functional testing by means of a novel test framework. Within the framework, the tests are automatically derived from a proposed new sort of requirements specification for communication services, the Service Description, and afterwards generated by means of predefined test modules. Finally, the test cases are executed against the System under Test, the communication service.**

Keywords—automated functional testing; communication services; requirements specification; test framework; testing methodology

I. INTRODUCTION

In the telecommunication domain, network operators and service providers aim for fast, easy and cost-efficient provisioning of value-added communication services. A fast transition from concept to market product and low price of new communication services is necessary due to the increasing competition in the telecommunication industry. The sum of these demands leads to reducing complete and sufficient functional testing which has a bad impact on the quality of the service. Moreover, functional testing procedures have to be executed consequently before the delivering of a communication service to a customer because the service provider has to assure that the communication service is executed properly and according to the specified requirements and that the communication service may not cause undesired behaviour within the provider's service environment.

In order to avoid these problems, the whole test development cycle specifically for communication services has to be improved. This starts with the requirements analysis which is usually standard Unified Modelling Language (UML) use case design [1; 2], with mostly natural language-based descriptions. This often leads to a lack of clarity as it is difficult to use language in a precise and unambiguous way. Besides the requirements analysis, the service testing is oftentimes manually done by test developers who gain their knowledge about a service's functionality from the natural language-based requirements specification. The test developer has to spend a significant amount of time on test case design, test data selection, and test evaluation.

In this paper, we propose a new test framework in order to do automatic functional testing of communication services. The foundation of the testing methodology is based on the definition of a new sort of requirements specification for communication services, the Service Description. After the Service Description is specified for a new communication service, it is parsed by a special test framework artifact which reads out the significant content and generates a formal behaviour model by composing predefined parameterised test modules. From the behaviour model, the functional test cases are derived and generated into executable TTCN-3 test cases which are subsequently executed against the communication service within a TTCN-3 test execution environment.

The remainder of this paper is structured as follows: the related work is presented in section 2. Section 3 introduces the novel architecture of the test framework and describes the testing methodology. Afterwards, section 4 discusses the demands on a new sort of requirements specifications for communication services and introduces the proposed Service Description. A simplified example of a communication service specification using the Service Description is discussed in section 5 and section 6 concludes the paper.

II. RELATED WORK

Our survey of the related literature shows that many different methodologies have been developed in the field of automated testing, mostly in the field of process-based systems. These approaches either generate test paths directly from code, based on data and control flow information [3; 4], or translate the code into formal specification languages like Petri nets [5],

[6], to perform the model checking and test derivation. A major disadvantage of these approaches is that the tests cannot detect the deviations from the functional specifications as the tests are directly derived from the code.

A series of methods is proposed in [7] to capture requirements and then manually transform them into conceptual models composed of object models (e.g. class diagrams), dynamic models (e.g. state machines and sequence diagrams) and functional diagrams. The authors introduce a set of techniques for users to precisely specify requirements and describe rules how the users can derive conceptual models from these requirements. The paper [7] does not mention a complete transformation method. Besides, the effort for a human to define both requirements and conceptual models seems to be very high.

An approach to generate finite state machines from use cases in restricted Natural Language (NL) is proposed in [8]. The approach needs the existence of a domain model which serves two purposes: a lexicon for the NL analysis of use cases, and the structural basis of the state transition graphs being generated. The domain model acts as the lexicon for NL analysis of the use cases because the model elements of the domain model are used to document the use cases. It is imaginable that an enormous user effort is needed to obtain a domain model containing classes, associations, and operations which are indispensable for generating state machines. There is no proof that the restricted NL is sufficient to describe the use cases.Yet no case study is presented to evaluate the approach.

In [9], the authors use a behaviour engineering methodology to formalise and validate a requirements specification and extend it with appropriate test activities. It is shown how testing information may be weaved into behaviour trees by identifying the system's boundaries and the definition of test action. The approach lacks the information of how tests are generated and executed but it seems necessary to transform the behaviour trees in state machines.

The generation of test cases from complete system requirements models is discussed in [10]. The model is described in a requirements specification language (SpecTRM-RL) which is based on a formal state machine model. Nevertheless, according to the authors, the notation is simple to read and understand for non-experts. However, the approach allows the definition of requirements models in different degrees of abstraction. Besides, the generation of tests and their execution is indeed discussed but not further defined.

A tool-based methodology to model-driven system testing of service-oriented systems is introduced in [11]. Additionally, it provides full traceability between the requirements, the system and test model. This aspect, however, leads to an amount of work for the human as the requirements have to be specified, the system model has to be defined as well as the test model. For the execution of test cases, the outdated technologies RMI (Remote Method Invocation) and CORBA (Common Object Request Broker Architecture) have been applied.

Whittle and Schumann proposed an approach [12] to automatically generate UML state machines from UML sequence diagrams. However, widespread modeling techniques like UML are too generic and lack the formalism required for domain modelling such as the requirements modeling of communication services.

Most approaches described in literature lack the definition of a testing methodology from the definition of the requirements until generation and subsequent evaluation of the functional tests. There is no discussed framework covering these steps specifically for communication services. Besides the standard way of defining requirements of communication services through UML use case design no further approaches are discusses so far.

III. TEST FRAMEWORK ARCHITECTURE

This section provides an overview of the underlying components and principles within the developed test framework.

Fig. 1 shows the artifacts of the test framework. The workflow of the testing methodology is triggered by a Test Developer whose role is the compilation of the Service Description. The Service Description is a new sort of requirements specification for communication services for the purpose of specification-based functional testing. It contains static architectural definitions describing the participating roles involved in the consumption of a communication service and their relationships as well as dynamic behavioural definitions specifying use-case related requirements on the part of the communication service. In the compilation phase, the Test Developer has to follow a well-defined guideline to define a Service Description for a communication service. Within the testing methodology this is the only task being carried out by a human, the subsequent process performs automatically. A more detailed introduction regarding the Service Description will follow in section 4.

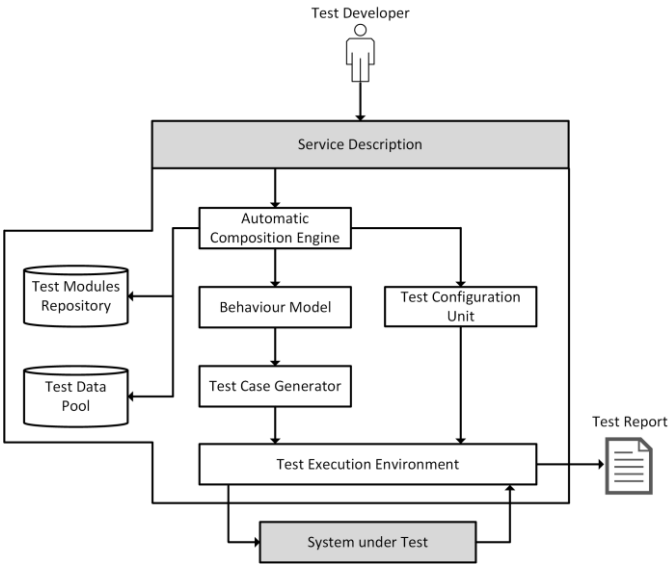

Fig. 1. Test framework artifacts and methodology

According to Fig. 1, the Service Description will be delivered to a very significant component within the testing

methodology, the Automatic Composition Engine (ACE). The main task of the ACE is the generation of a system model, the Behaviour Model, which is a complete formal model or rather Extended Finite State Machine (EFSM) describing the desired and possible behaviour of the specified communication service.

In order to generate the Behaviour Model, the ACE first parses the architectural definitions from the Service Description and forwards them to the Test Configuration Unit (TCU). The TCU thereupon extracts the relevant information for the Test Execution Environment (TEE) such as the System under Test (SUT) addressability, the participating test components and the data structures being exchanged between SUT and test system.

The ACE parses the behavioural part of the Service Description and identifies the participating roles within the specified requirements to select suitable test modules from the so-called Test Modules Repository (TMR). The TMR is a database containing predefined modular EFSMs, so-called test modules, which cover typical communication service characteristics such as sequences of multimedia protocols like SIP (Session Initiation Protocol) or RTP (Real-Time Transport Protocol) and other important protocols, e.g. HTTP (Hypertext Transfer Protocol). The test modules usually define a protocol-specific behaviour of a certain use case, e.g. the sending of an instant message by using the SIP protocol, and cover both standard behaviour as well as possible alternative behaviour like timeouts. To sum it up, the test modules define the standard compliant behaviour of a certain use case. Additionally, the test modules are parameterised in order to configure the test data.

After selecting the appropriate test modules from the TMR, the ACE connects to the Test Data Pool, a database containing collections of test data templates for each test module within the TMR. Then the ACE chooses the adequate test data templates and the parameters from the Service Description are included. After that, the ACE starts with the composition of the test modules. Each test module has interfaces which are linked to the existing states within the underlying EFSM of a test module. If two test modules are to be combined, the originating test module and the destination test module are connected with a transition between their interfaces. The task of the ACE is to realise the connection according to the use-case related information within the Service Description. Obviously, not all the interfaces within one test module has to be operated, it depends on the descriptions. However, at least the interfaces within the start state and the end state of a test module have to be activated excepting the first and the last test module to be composed.

After the composition of the chosen test modules is fulfilled the dependencies of the parameterisations for each test module have to be dissolved. This is necessary in order to reuse and change parameter values being defined in one test module for the other test modules within the composed model. This is important if certain parameter values defined in test module A have to be reused in test module B. This could be for instance a SIP URI which was defined in A and has to be reused in B.

As mentioned before, the result of the composition is the Behaviour Model which is then delivered to the Test Case Generator (TCG). The TCG contains a test case finder which uses an algorithm to enable the derivation of abstract test cases from the Behaviour Model. This algorithm optimises the traversal of the EFSM by combining depth-first search and breadth-first search. After the extraction of the abstract test cases, a test code generator translates them into executable test cases by means of a special mapping concept which is described in [13]. The executable test cases are defined in TTCN-3 (Testing and Test Control Notation), a test scripting language which was standardised by ETSI [14] and ITU-T [15; 16], and supports the modularised creation of test scenarios for message and procedure based systems.

The final step of the methodology takes place within the TEE which receives both the executable TTCN-3 test cases from the TCG and the relevant information about the SUT and the participating test components from the TCU. Based on the information, the TEE selects the appropriate system adapter and codecs. The system adapter [17] adapts the communication of the TTCN-3 test system to the specific execution platform of the SUT whereas the codec is responsible for the encoding and decoding of TTCN-3 values into bitstrings so that the data can be sent to the SUT. Finally, the system adapter and codecs are added to the Test Suite and the generated TTCN-3 test cases are executed against the SUT. A test log is written which documents the test case execution and the reactions of the SUT. The data is formatted and integrated into a Test Report to demonstrate if all the tests were successful due to the defined requirements specified in the Service Description. If there are mismatches the whole process has to be verified, the SUT as well as the Service Description.

IV. Service Description

A well-defined requirements specification is the critical component when it comes to functional testing as it represents the foundation for every derived test case. Especially with reference to the proposed test framework introduced in the last section, several demands on the Service Description were discussed.

A. Demands on the Service Description

The Service Description should meet some general demands which are relevant for any kind of specification document. First of all, the Service Description should be complete and has to contain all the requirements which describe exactly the desired behaviour of a communication service. The specified requirements should be understandable and not ambiguous. The Service Description should not contain any contradictions and changes can be done without difficulties. It should be machine-readable and interpretable so that the ACE is able to parse the content.

Besides the general demands, the proposed test framework requires some further specific demands with reference to the artifacts within the testing methodology. The ACE for instance requires the description of behavioural aspects which can be described in terms of use-case related requirements. Each requirement has to be traceable throughout the whole testing process from its definition within the Service Description by the Test Developer until the execution of the automatically

derived test cases. Therefore, a formal semantic relationship between the requirements and test cases has to exist. Also, the requirements have to contain information about the participating roles so that the ACE can select the appropriate test modules while parsing the Service Description. As the requirements describe the specification of a communication service they address a subset of the protocol-specific behaviour defined in the test modules. Possible relations and dependencies between requirements can lead to compositions of test modules. Another important demand on the Service Description is the support for applying the test data from the Test Data Pool. Within the requirements it should be easily possible to parameterise and address the test data sets.

Finally, with reference to the test configuration within the Test Execution Environment, the Service Description has to contain all the relevant information about the test architecture, which is a representation of the structural aspects of the test system, such as SUT information, test components and required codecs.

B. Service Description structure

As mentioned in section 3, the proposed Service Description is subdivided into architectural and behavioural definitions. Besides, some further information is given. Fig. 2 illustrates the structure of the Service Description.

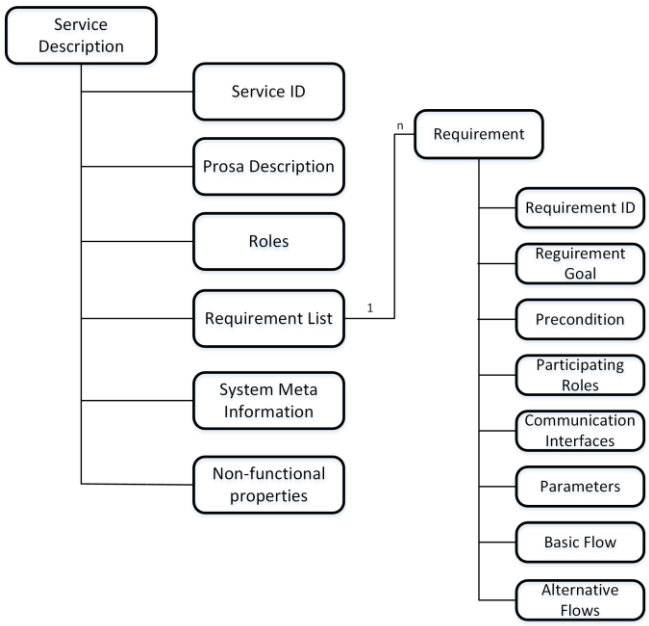

Fig. 2. Structure of the Service Description

The *Service Description* element is the root of every instance of a requirements document. It contains the *Service ID*, a unique identifier for the communication service to be specified. This is an important attribute as it determines the name of the test suite to be generated. The *Prosa Description* contains an abstract description of the communication service's functionality. In the *Roles* attribute, all the participating users who consume the communication service are listed. Roles could be for instance Web Browsers or SIP Softphones. The definition of the roles is the basis for the selection of the test components within the test configuration. Further test configuration properties are defined in the *System Meta Information* containing SUT information in order to build up the test configuration. Here, the service addressability is defined such as the service URI, IP addresses and port numbers. A predefined variable list is available to assure that the relevant parameters are set. The *Non-functional properties* contain non-functional requirements like costs.

A further important part of the Service Description is the *Requirement List* which defines the behavioural part and contains all the relevant requirements a communication service has to fulfill. The specification of each requirement in the *Requirement List* is well-defined by the following components in Table I.

TABLE I. SPECIFICATION OF A REQUIREMENT

Component	Description
Requirement ID	Unique identifier for a requirement.
Requirement Goal	Prosa description of the requirement's target.
Precondition	Determination of depending flows within other requirements that have to be carried out before the Basic Flow of this requirement can take place.
Participating Roles	List of the roles involved in this requirement.
Communication Interfaces	Definition of the relevant system side communication interfaces.
Parameters	Determination of the required parameters within this requirement.
Basic Flow	Description of the steps that have to be taken to achieve the target of the requirement.
Alternative Flows	Description of exceptional behaviour. Each step within a Basic Flow can lead to an Alternative Flow.

The significant part of a requirement is the use case description of the *Basic Flows* and *Alternative Flows*. In standard textual UML use case design, natural language-based descriptions are used. There are many documented approaches [18], where restriction rules for textual use case design are applied to reduce the imprecision and incompleteness. However, even if restricted vocabulary is used, formulation oftentimes is confusing and error-prone. The larger a requirements specification is the more problems arise disproportionately with natural language-based specifications. With reference to the testing methodology, the *Basic Flows* and *Alternative Flows*, the descriptions should be machine-readable so that the ACE can parse and interpret them. Therefore, a new formal approach is required which enables the description of behaviour flows and realises the reference to the test modules within the TMR and the test data.

As an appropriate formal method the usage of a process algebra notation has been found, the pi-calculus [19]. In general, the pi-calculus is a simple language to specify interactive message-passing programs. It provides mathematical foundations of some modern workflow languages like the Business Process Execution Language (BPEL) and is more concise than automata, very expressive

and even easier to develop. However, the pi-calculus is so minimal that it does not contain primitives such as numbers, booleans, variables, functions, or even usual flow control statements such as if-then-else constructions. The syntax just consists of a set of prefixes and process expressions which is illustrated in Table II.

TABLE II. BASIC PI-CALCULUS SYNTAX

Syntax	Description
P ::= 0	Process P is a null process.
P | Q	Parallel composition of processes P and Q.
!P	Replication of process P.
´a<x>.P	x is sent along channel a, then process P starts.
a(x).P	Channel a receives x, then process P starts.

The mentioned limitations of pi-calculus may justify why it has not been applied as a requirements specification language for functional testing methodologies so far. Therefore, we propose an applied pi-calculus language in order to describe the *Basic Flow* and *Alternative Flows* within the requirements properly. The conceptual idea was derived from [20]. In that approach the grammar for processes is similar to the one in the pi-calculus, except that messages can contain terms. In our proposed pi-calculus, we reuse the ideas of terms to define flow control statements, variable usage and method invocation. Furthermore, we reuse the channels to express possible outputs and inputs on the part of the system side communication interfaces.

Our proposed enhancements of the pi-calculus syntax are illustrated in Table III.

TABLE III. ENHANCED PI-CALCULUS SYNTAX

Syntax	Description
if x == {value} then P else Q	If the variable x contains the value the process P starts otherwise process Q.
´a<httpServer→response (200)>	Through channel a, a 200 response is sent from the communication interface httpServer.
response→statusCode=200	The attribute statusCode of the complex variable response is set to the value 200.

The complex variables such as *response* within the description of Table III are parameters which can be loaded from the Test Data Pool. The arrow symbolises the access to the attributes of the complex data structure. In standard programming languages this would be the dot operator. As the dot has a different meaning in pi-calculus, namely the separation of process steps, the arrow is used in our approach.

With reference to the description of the requirements, each *Basic Flow* and each *Alternative Flow* can be defined by one pi-calculus process in the Service Description. Again, each process contains n channels where each is representing the communication between the communication service and the components which are depending on the defined communication interfaces. The mentioned enhancements of the pi-calculus show that certain conditions can lead to different behaviour which is specified through different following processes.

In the following, an example of a communication service specification with the Service Description will be described.

V. EXAMPLE

The example communication service being specified by the Service Description is called Click-2-Instant-Message. The service flow starts with a text message and a destination SIP URI being typed in by a user on a website. By actuating a button the message is sent via HTTP protocol to an application server with the deployed Click-2-Instant-Message service. Subsequently, the service sends a SIP Message containing the text message from the website to the SIP phone with the stated SIP URI.

The architectural part of the Service Description enables the building of the test configuration and is illustrated in Table IV.

TABLE IV. EXAMPLE SERVICE DESCRIPTION ARCHITECTURAL PART

Service ID	Click-2-Instant-Message
Prosa 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 message. 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 no text was typed, the user should be informed with "No text input" on the website. If the SIP URI was invalid the user should be informed with "No valid SIP URI". If the transfer worked, a success message should occur, "Message sent successfully".
Roles	Web Browser, SIP Softphone
System Meta Information	ServiceURI: sip:click2IM@sip.de
Non-functional properties	None

The defined roles leads to the fact that two test components are required with one understanding the HTTP protocol (Web Browser) and the other one understanding the SIP protocol (Softphone). The SUT is reachable through the ServiceURI which is specified in the *System Meta Information*.

Furthermore, the behavioural part of the Click-2-Instant-Message Service Description contains one requirement. The requirement determines amongst others the communication interfaces which describe the communication channels from the SUT to the test components. The declaration of the communication interfaces automatically leads to the selection of the test modules, in this example *HTTP_Server* and the *SIP_UAClient_MESSAGE*. A more detailed description of the structure of the test modules is described in [21]. Every test module contains a set of parameters. The relevant ones for the specified flows in the requirements have to be determined like in Table V.

TABLE V. EXAMPLE SERVICE DESCRIPTION BEHAVIOURAL PART

Requirement ID	**1**
Requirement Goal	Initiator wants to send a text message from a website to a SIP softphone.
Precondition	None
Participating Roles	Web Browser, SIP Softphone
Communication Interfaces	HTTP_Server [w] → channel **a** SIP_UAClient_MESSAGE [s] → channel **b**
Parameters	[w]→httpRequest; [w]→httpResponse; [s]→sipRequestMessage; [s]→sipResponse2xx_6xx
Basic Flow	P ::= a([w]→httpRequest(text, targetURI)). **if** text == NULL **then** Q **else**. **if** isValidURI(targetURI) == false **then** R **else**. ´b<[s]→sipRequestMessage(targetURI, text)>. b([s]→sipResponse2xx_6xx(200)). ´a<[w]→httpResponse(200, "Message sent successfully")>.0
Alternative Flow 1	Q ::= ´a<[w]→httpResponse(200, "No text input")>.0
Alternative Flow 2	R ::= ´a<[w]→httpResponse(200, "No valid SIP URI")>.0

The behavioural flows are described in the proposed pi-calculus syntax. The Basic Flow specifies the process *P* with an incoming HTTP request over the channel *a* containing the parameters *text* and *targetURI*. Then the content of both parameters is checked. If *text* does not have content, process *Q* is triggered, otherwise if *targetURI* contains an invalid SIP URI, process R is triggered. If both parameters are correct, a SIP Message with the *text* is expected to be sent over the SIP channel *b* and acknowledged. At the end, a HTTP response is sent over channel *a* to inform that the transfer was successful. The sum of flow descriptions in this example define a specification of a service which describes a certain subset of the flows being contained in the test modules.

VI. CONCLUSION

Automated functional testing of communication services directly from a requirements specification requires its understandability, completeness and machine-readability. Such a requirements specification, the Service Description, was introduced and exemplified in this paper. Besides, its significance was discussed with reference to the proposed test framework.

The presented approach empowers network operator and service providers to deliver high quality communication services in a cost and time optimised way to their customers. The services undergo a continuous testing procedure based on a new functional testing methodology.

REFERENCES

[1] O. Ryndina, P. Kritzinger, "Improving requirements specification for communication services with formalised use case models", Proceedings of the Southern African Telecommunication Networks and Applications Conference (SATNAC 2004), Spier Wine Estate, Western Cape, South Africa, September 2004.

[2] A. Eberlein, M. Crowther, F. Halsall, "Development of new telecommunications services using an expert system", BT Technology Journal, vol. 15, pp. 2137-222, 1997.

[3] J. Yan, Z. Li, Y. Yuan, W. Sun, and J. Zhang, "BPEL4WS Unit Testing: Test Case Generation Using a Concurrent Path Analysis Approach", Proceedings of the Seventeenth International Symposium on Software Reliability Engineering (ISSRE 2006), Raleigh, North Carolina, USA, November 2006.

[4] Y. Yuan, Z. Li, and W. Sun, "A graph-search based approach to bpel4ws test generation", Proceedings of the International Conference on Software Engineering Advances, Papeete, Tahiti, French Polynesia, October 2006.

[5] J. Garcia-Fanjul, J. Tuya, and C. de la Riva, "Generating test cases specifications for BPEL compositions of web services using SPIN", Proceedings of the International Workshop on Web Services: Modeling and Testing, Palermo, Italy, June 2006.

[6] Y. Zheng, J. Zhou, and P. Krause, "A model checking based test case generation framework for web services", Proceedings of the International Conference on Information Technology (ITNG 2007), Las Vegas, Nevada, USA, April 2007.

[7] E. Insfrán, O. Pastor, R. Wieringa, "Requirements Engineering-Based Conceptual Modelling", Requirements Engineering Journal, vol. 7, pp. 61-72, June 2002.

[8] S. Somé, "An approach for the synthesis of state transition graphs from use cases", CSREA Press, vol. 1, pp. 456-462, 2003.

[9] M.-F. Wendland, I. Schieferdecker, A. Vouffo-Feudjio, "Requirements-driven testing with behaviour trees", Proceedings of the Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW 11), Berlin, Germany, March 2013.

[10] K. Kelley, "Automated Test Case Generation from Correct and Complete System Requirements Models", Proceedings of the IEEE Aerospace Conference, Big Sky, Montana, USA, March 2009.

[11] M. Felderer, P. Zech, F. Fiedler, R. Breu, "A Tool-based methodology for System Testing of Service-Oriented Systems", Proceedings of the Second International Conference on Advances in System Testing and Validation Lifecycle (VALID 2010), Nice, France, August 2010.

[12] J. Whittle, J. Schumann, "Generating statechart designs from scenarios", Proceedings of the Twentysecond International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, June 2000.

[13] P. Wacht, T. Eichelmann, A. Lehmann, and U. Trick, "A New Approach to Design Graphically Functional Tests for Communication Services", Proceedings of the Fourth IFIP International Conference on New Technologies, Mobility and Security (NTMS 2011), Paris, France, February 2011.

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

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

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

[17] S. Blom et al., "TTCN-3 for Distributed Testing Embedded Software", Proceedings of the Sixth International Andrei Ershov Memorial Conference on Perspectives of Systems Informatics (PSI 2006), Berlin, Germany, 2007.

[18] T. Yue, S. Ali, L. Briand, "Automated Transition from Use Cases to UML State Machines to Support State-based Testing", Proceedings of the Seventh European Conference on Modelling Foundations and Applications (ECMFA 2011), Birmingham, United Kingdom, June 2011.

[19] D. Sangiorgi, "The Pi-Calculus: A Theory of Mobile Processes", Cambridge University Press, 2008.

[20] M. Abadi, C. Fournet, "Mobile Values, New Names, and Secure Communication", Proceedings of the Twentyeighth ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, United Kingdom, January 2001

[21] P.Wacht, T. Eichelmann, A. Lehmann, U. Trick, "A New Approach to Design Graphically Functional Tests for Communication Services", Proceedings of the Fourth IFIP International Conference on New Technologies, Mobility and Security (NTMS 2011), Paris, France, February 2011