Automated Functional Testing of P2P-based M2M Applications

Besfort Shala, Patrick Wacht, Ulrich Trick, Armin Lehmann
Research Group for Telecommunication Networks
Frankfurt University of Applied Sciences
Frankfurt/M., Germany
shala@e-technik.org

Besfort Shala, Bogdan Ghita, Stavros Shiaeles
Centre for Security
Communications and Network Research
University of Plymouth
Plymouth, UK

***Abstract*—This publication proposes a novel testing framework for the functional verification of decentralised services and applications in Peer-to-Peer (P2P)-connected Machine-to-Machine (M2M) networks. Besides presented challenges and requirements for testing within distributed environments, a concept for deriving and generating test cases based on a Test Description Language (TDL)-based approach is described and mapped to M2M applications.**

***Keywords*— M2M; P2P; Service and Application; Functional automated testing**

I. Introduction

Building surveillance, energy management, traffic management, electro mobility and ambient assisted living are only a few Machine-to-Machine (M2M) application fields which are present nowadays. According to the European Telecommunications Standards Institute (ETSI), M2M applications are defined as "applications that run the service logic and use Service Capabilities accessible via open interfaces" [1]. Previous papers have defined requirements and concepts to realise service and application provision in M2M. The work and investigations of this research paper are based on the P2P-based M2M application (P2P4M2M) framework which offers new possibilities for applications, realised by several peers, independent of central instances or corporations [2].

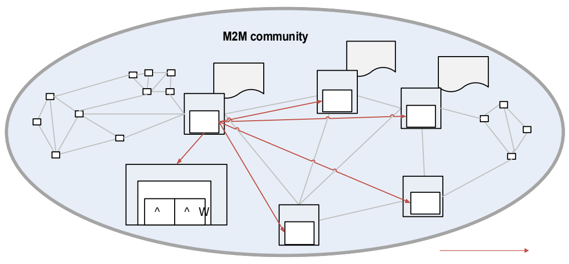

Fig. 1. P2P connected peers within a M2M community [2]

Reference [2] defines a framework that realises service and application provisioning using P2P networking in M2M application field. An application consists of one or more underlying services that are combined (i.e. aggregated or composed). Also, the use of the community concept described in [2] helps to avoid legal restrictions, adjust different interests among the peers and ensure optimisation and forming P2P networks. Fig. 1 shows the structure of the P2P connected peers within an M2M community based on [2]. Besides many advantages of the service provisioning concept, [2] does not consider approaches for testing P2P-based services/ applications in M2M. Therefore, a novel testing framework is required to enable testing of heterogeneous and decentralised services and applications in the P2P4M2M framework.

The aim of this paper is to illustrate the challenges and requirements of testing services and applications in P2P4M2M and to define a novel concept for functional automated testing. Through a functional testing approach, it can be verified that services and applications are running properly and according to the user's requirements. For dealing with the distributed nature of services and applications in [2], this paper introduces a novel testing framework with a special testing architecture. The proposed testing architecture integrates a Test Generation Environment.

To show the importance of this research work, the following paper is structured into seven sections. After the introduction, section II presents an overview about the concept of service and application provisioning based on the P2P4M2M framework. Section III illustrates related work on testing approaches. Testing challenges and requirements are presented in section IV. Section V gives an overview about the principles of the proposed testing framework and describes the testing architecture and its elements. Section VI shows the test derivation and generation concept. At the end, section VII concludes with an application example related to the testing concept presented in this research.

II. P2P-based M2M Applications

Reference [2] presents a concept of service and application provision in M2M. A service, as well as an application, can be realised by peers using technical or non-technical principles

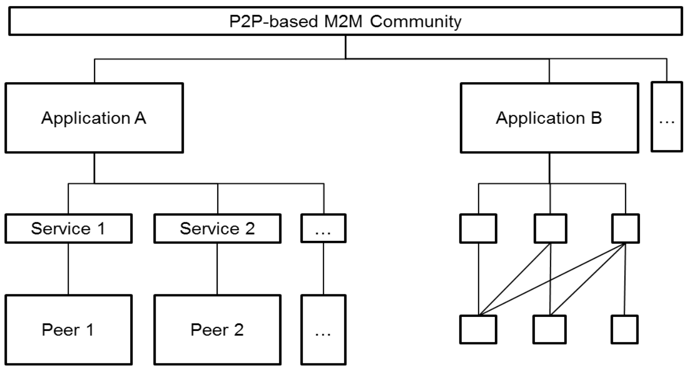

Fig. 2. Generalised structure of P2P4M2M framework [2]

(i.e. it can be provided using technical devices, e.g. computers, or by a human, e.g. personal assistance services). The services are realised by one or more service components which form the building blocks of services. The service components themselves are realised via several software applications executed on several execution environments. Peers are also represented by technical devices or humans (if applicable supported by technical devices) which are networked using P2P mechanism. The M2M community described in [2] forms a social network of peers where different sub communities are also used to address different application fields, interests and geographical locations. The networking enables the participating peers to provide a service that can be consumed by others [2].

Reference [3] introduces a Service Management Framework (SMF) installed in the local households, consisting of Service Delivery Platform (SDP) and Service Creation Environment (SCE), and uses the concept of P2P networked energy-community. The SCE brings the functionality to design and configure value-added services graphically, according to the personal needs of the users [3]. The SMF described in [3] is also the main component for service and application provisioning in M2M based on the P2P4M2M framework [2].

According to [4], the information exchange between the peers for the service utilisation and the signalling to generate the application is enabled by using various M2M communication protocols (e.g., CoAP, HTTP, SIP, MQTT) based on SUBSCRIBE/ NOTIFY principle. Fig. 2 shows a generalised overview of the P2P4M2M framework mentioned in [2].

III. Related work

To the author`s knowledge, there are no known studies about automated testing and securing services and applications within the P2P4M2M framework. Furthermore, only a very limited amount of related work exists on functional testing of P2P systems and M2M applications.

It is crucial to define a suitable testing architecture for testing different services and applications based on the P2P4M2M framework. Our survey of the related literature shows some centralised approaches for testing distributed systems. Several publications [5-8] present testing architectures based on a coordinator and testers with focus on testing P2P functions and distributed systems. The coordinator inserts and controls several testers which run on different logical nodes. Also, the coordinator collects centrally information of the distributed System under Test (SUT), derives the test verdicts, observes and controls external and internal actions of SUT and has a global view on distributed SUT. The testers execute the test instructions received from the coordinator and control the volatility of single peers. The problems of these approaches are the single points of failure of the global tester and the low level of usability in large scale systems as they do not scale up to large numbers of peers (typical P2P system may have a high number of peers). Another problem is the non-efficient generation of test cases and the missing environment for test generation. There are also decentralised approaches for testing distributed systems such as in [6] who introduces distributed testers with the following functions: several operating tester components which process a global test case together. The behaviour of the testers is controlled by a test coordination procedure. Reference [9] also proposes a distributed test architecture without the use of a central coordinator and ensuring the coordination between the testers by introducing a distribution procedure of test sequences among the testers. The disadvantage of this approach is that the testers do not have a global view on the SUT, so they must synchronise each other by means of a test coordination which leads to a high effort for synchronisation events for coordination between the testers.

Besides the fundamentals of the test architecture, the derivation of test cases needs to be analysed. In literature, many model-based approaches are identified, e.g. in [10]. Here, one or more formal models are created from which test cases can be automatically generated and executed according to predetermined test criteria. There are different notations which can be applied to model-based testing such as Statecharts [11], Finite State Machines (FSM) [12], Petri nets [13] and UML [14]. Reference [15] presents an automated functional testing approach based on Statecharts notation for modelling the potential behaviour of a service. However, the approach leads to an enormous amount of generated test cases and is not applicable for distributed systems such as the P2P4M2M framework. A promising concept has been developed by ETSI, the Test Description Language (TDL) [16]. The language TDL is scenario-based and allows the design, documentation and representation of formal test descriptions. It already includes the necessary components for automated test design such as test data, test configuration, test behaviour and test objectives.

IV. Challenges and requirements for testing applications in P2P4M2M

According to [17], testing is defined as "the process of analysing a software item to detect the differences between existing and required conditions and to evaluate the features of the software items". The aim of this research is the testing of services and applications based on the P2P4M2M framework. The process of creating M2M applications based on [4] makes functional testing very complex and can be described as follows: The application creator creates and configures an M2M application using his SCE. The M2M application consists of several services which are part of an M2M community. The services are described by their Service Interface Description (SID). The SID includes service ID, service functions, input, output and further configuration parameters of a service. Services are provided by different peers participating in a P2P network without the use of a central authority. The creation of an application will generate an SCXML (State Chart XML) description which precisely describes the potential functionality the application should deliver in a formal manner. In principle, such an SCXML description includes the involved services, the connection of services as well as conditions and definitions of input/ output parameters.

A special testing framework is required for testing the M2M application. Firstly, the P2P4M2M framework is a distributed system and based on [18] distributed systems are "heterogeneous in terms of communication networks, operating systems, hardware platforms and also the programming language used to develop individual components". Reference [18] states that the size and complexity of distributed systems is growing and the system should be able to run over a wide variety of different platforms and access different kinds of interfaces. Considering the distributed characteristics of the P2P4M2M framework and the need for exchanging relevant information for testing it is important to ensure collaboration between the services, applications and test elements. The decentralisation of the peers and their volatility (nodes leaving and entering suddenly) in the P2P-based M2M application community should be considered in the test environment. Especially in the P2P4M2M concept, the application creator could be a user who has no technical background and who is not able to prepare the testing. Also, based on [10] the application creator should not be the test creator. If the application creator has already interpreted a specification incorrectly during the development, he will also misinterpret it for the test. For this reason and for the advantages of test automation [10] the testing needs to be automated using a mechanism which utilises the information provided by the application and the services. Another problem is the procedure for defining test cases. Testing distributed services and applications in M2M networks requires different methods for deriving and generating test suites and for running the test. Reference [19] presents several problems for testing distributed systems including the test data generation and the specific execution behaviour. The testing framework must have the ability to derive test cases from the information gathered by the application and the distributed services. Based on the characteristics of the P2P4M2M framework presented in [4] the execution of the test cases on the participating services and the composed application should also be considered. Due to these different challenges, the general requirements for the testing framework can be summarised as follows:

- **Collaboration** – It is necessary to have collaboration between the application creator, the test environment and the peers, which are part of the application, and are providing or consuming services.
- **Deployment** – The testing framework needs to have the ability to deal with high number of peers and also the volatility of nodes in P2P network should be considered by the framework.
- **Test Automation** – Based on the complexity of the P2P4M2M framework the whole testing process needs to be automated considering the distributed architecture of the system.
- **Test Derivation** – Test suites need to be derived and generated from the gathered information about the composed M2M application and the participating services.
- **Test Execution** – The generated test cases need to be executed on different services in a timely manner. Also, the test cases for the whole application should be executed after its creation.
- **Verification** – The testing process should deliver results about the functionality of the considered SUT, which could be a service or an application.
- **Tool support** – The framework should provide tools to generate, execute and manage tests.
- **P2P and M2M capability** – The framework should consider the included P2P mechanism and its characteristics within the application framework. M2M communication protocols should also be supported.

V. PROPOSED TESTING FRAMEWORK

The challenges of testing (see section IV) and the complexity of the P2P-based M2M application framework leads to the necessity to define a suitable testing framework. The focus within this research work is the functional testing of services/ applications based on the P2P4M2M framework. Functional testing is the process of verifying the functions in a system to assure that they meet the specified requirement. Reference [20] defines that "every software system can be seen as a black box, where a tester selects valid and invalid inputs and determines the correct output" and in functional testing "a tester does not need to know the internals of the SUT as the focus is to evaluate the functional correctness of a given system, independently of its internal implementation".

Two black-box testing scenarios can be derived based on the application creation process described in [4]. The first scenario deals with the testing of a service after it enters the M2M community. This happens to ensure the availability of the correctly working services in the community and should be done after predefined time intervals. The second testing scenario will happen after the application creator builds an application using several services participating in the M2M community. The composed and created application needs to be tested based on its configuration and according to the special conditions of each participating service in the composition. An example for testing an application is provided later in this research paper. The services, which are part of the composed application, need to be tested according to their special configurations within the application.

Based on the related work [5-9] [14-15], the requirements for the testing framework for P2P-based M2M applications and the need for a load balanced and effective testing mechanism, a test architecture with a combination of a global tester called Test Master and distributed testers called Test Agents is presented in this work. Therefore, a test generation environment is included in the testing framework which derives and generates test cases and interacts with the Test Master, the services and the application creator. Fig. 3 shows the conceptual test architecture consisting of a Test Master, Test Agents and Test Generation Environment (TGE).

In the approach, the TGE receives an SCXML description of a composed application from the Service Creation Environment (SCE). The major role of the TGE is to derive and generate adequate test cases and to transfer these as test instructions to the Test Master. In the further step, the Test Master controls and manages the test process based on the test instructions. Accordingly, the Test Master sends test instructions to the Test Agents which will afterwards execute the test cases on composed services (the application) representing the System Under Test (SUT). After the test process terminates, the participating Test Agents send their test results (including the test verdicts) to the Test Master which will generate an overall verdict for the application.

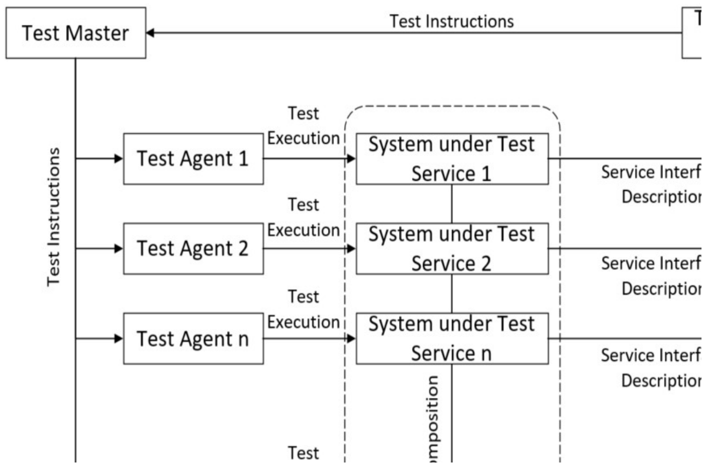
Test Master
Test Instructions
Test Instructions
Test Execution
Test Agent 1
System under Test Service 1
Service Interf Descriptior
Test Execution
Test Agent 2
System under Test Service 2
Service Interf Descriptior
Test Execution
Test Agent n
System under Test Service n
Service Interf Descriptior
Test
mposition

Fig. 3. Conceptual Test Architecture of the P2P4M2M framework

To establish the described test process, adequate test cases should be derived by the TGE. As the test cases prove the functionality of the composed services, they must be derived from a system model. Reference [21] defines system models as a tool for describing the composition and interaction of components in the system. System models are useful for testing because they provide general information about the functionality of a system. In this approach, an SCXML application description and the Service Interface Descriptions (SID) of the participating services serve as system model or rather specification notations. Both notations are machine-readable and can therefore be operated by the TGE.

As illustrated in the test process, the SCXML application description will be automatically sent to the TGE as soon as a new application has been created within the SCE. Principally, the application can be provided by all participating peers that provide the single services which taking part in the application. Therefore, the TGE first needs to detect all the potential combinations of peers to provide the application (see Fig. 4). In the P2P4M2M framework, each single service provided by a peer is determined by a unique identifier. As soon as a service is part of an application, its unique identifier needs to be included as attribute of a selected <state> element within the SCXML application description.

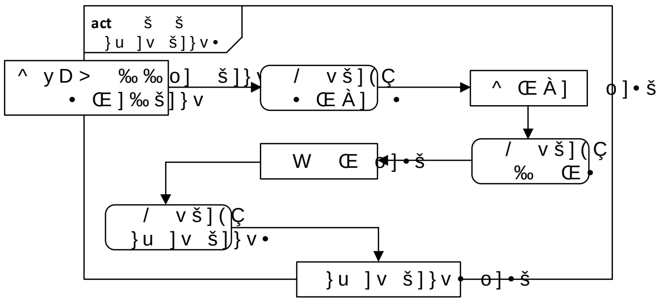

Fig. 4. Activity "Detect combinations"

After the TGE identified all collaborating services ("Service list"), it needs to figure out which peers are currently providing the services. Finally, all possible combinations of peers providing the services are generated and output of the activity "Detect combinations".

In a further step, the SIDs of the participating peers for their providing services becomes relevant. The TGE requests the SIDs from the peers and analyses them. One part of the SID is the test configuration which has been derived from the concept of the TDL-based test configuration. It consists of so-called tester and SUT components as well as their interconnections represented as connections (see Fig. 5).

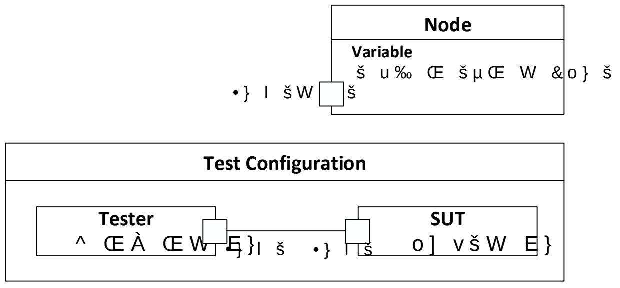

Fig. 5. Test configuration as part of example SID

The illustration shows a test configuration with one SUT component acting as client and one tester component acting as server. Both components are connected via their sockets and it is also specified which data (here a temperature value) is exchanged between the components. This example test configuration is simplified. In principal, the consumption of a service can involve more than one tester components in diverse roles. The test configuration is required to schedule the roles of the Test Agents in the conceptual test architecture (see Fig. 3) during test execution. A Test Agent can include one or more tester components that interact with the SUT. A further positive aspect of the TDL-based test configuration is the possibility to directly derive Testing and Test Control Notation 3 (TTCN-3)-based test configurations.

Besides the test configuration, further information is required by the TGE to generate test suites. Fig. 6 shows the activity "Generate test suites". Based on the combinations list which has been the output of the activity "Detect combinations" (see Fig. 4), the TGE requests the SIDs and loads the appropriate TDL-based test configurations (as shown in Fig. 5).

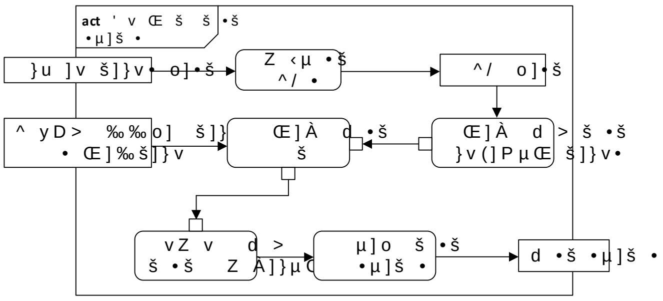

Fig. 6. Activity “Generate test suites”

In a next step, the test data is derived by means of the SCXML application description. The description includes a specific use case of the service and contains required parameters being used. Based on the parameter values (e.g. temperature values for sensors), varieties of threshold values can be generated by the TGE. Afterwards, one of the most important steps takes place, the determination of the test behaviour. In principal, the TDL-based test behaviour defines the expected behaviour of an SUT. It includes actions and interactions and can specify behaviour in an alternative, parallel, iterative and conditional way. There is also a possibility to specify defaulting, interrupting and breaking. For each service specified by an SID, the principal TDL-based test behaviour is determined (see Fig. 7).

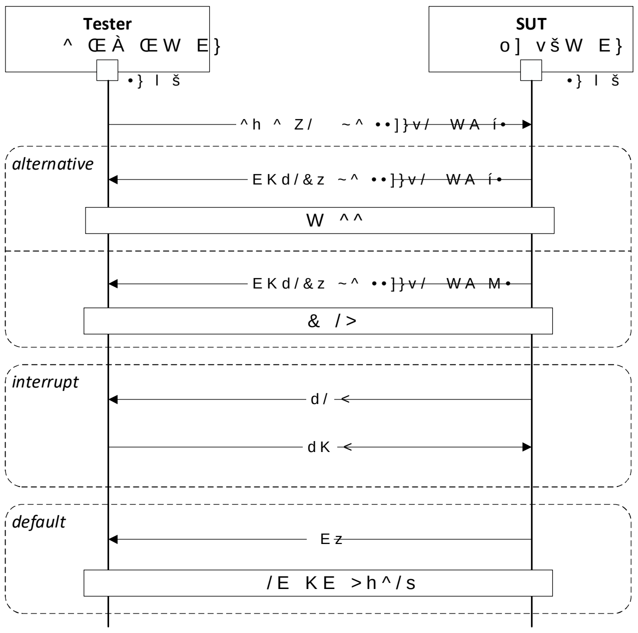

Fig. 7. TDL-based example test behaviour description

It is important that the test behaviour description includes all tester components specified in the TDL-based test configuration (see Fig. 5). In the illustrated example (see Fig. 7), alternative, interrupted and default behaviour is determined in the specification. It starts with the tester component sending a “SUBSCRIBE” message to the SUT component. Based on this trigger, the tester component expects a response from the SUT. In the alternative cases, a valid “NOTIFY” is expected which leads to a “pass” verdict. Here, it is also possible that the “NOTIFY” message includes invalid content. Maybe there is wrong data included or the session identifiers mismatch. Such a case would lead to a “fail” verdict of a test case. Besides the alternative cases, interruptions can be specified. Such messages have no effect on the functional purpose of the test case. So, no verdict will be given due to interruption messages. Finally, the default behaviour specifies messages from the SUT which do not fit or have a different purpose. In this case, the verdict will be “inconclusive”.

In the final step of the “Generate test suites” activity (see Fig. 6), all TDL-based test behaviour specifications are combined based on the participating services which are involved in the application.

VI. Application Example

To prove the concept of testing P2P-based M2M applications provided by the P2P4M2M framework, an example application is introduced in the following. The major idea behind the application called “Temperature Surveillance” is to allow consumers to continuously get informed about the temperature in certain rooms, e.g. via their smartphones. The application requires the involvement of three different services which exchange information by using SIP SUBSCRIBE and NOTIFY messages. Service 1 delivers the temperature values by accessing diverse temperature sensors in the environment whereas service 2 evaluates the received values from service 1 and determines the consumers (via SIP URIs) who should receive the values. The role of service 3 is to forward the received temperature values via SIP instant messages to the consumers. Based on this example, the proposed TGE generates a test execution architecture composed of one so-called Master Test Component (MTC) and several Parallel Test Components (PTC). Both the terms MTC and PTC are derived from typical TTCN-3-based environments [22]. The TTCN-3 concept also allows to define a distributed test execution environment where all test components are running on different machines. MTC and PTCs are the corresponding Test Master and Test Agents presented in section V. This aspect is very relevant for testing distributed applications running in the P2P4M2M framework. Especially for the “Temperature Surveillance” application, four PTCs are used (see Fig. 8). The number of PTCs depends on the TDL-based test configurations included in the SIDs of the services that are involved to provide the application functionality. For both services 1 and 2, it is sufficient to let only one PTC (PTC 1 for service 1, PTC 2 for service 2) participate in the test execution. For service 3, a random number of PTCs is required as the number of consumers is depending on the number of registered SIP URIs. An example test case verifying the main functionality of the application would be executed as follows: First, PTC 1 subscribes service 1 to continuously receive temperature values via NOTIFY messages. As soon as PTC 1 receives new values, they are forwarded to all the other PTCs via the MTC to compare them later in the test execution. Second, PTC 2 subscribes to service 2 to receive the temperature values and the SIP URIs of the consumers. PTC 2 will also verify if the temperature values received by PTC 1 match with the ones

PTC 2 received from service 2. If the values are not equal, the test case will be directly declared as "fail". If the values are equal, the test case execution continues with service 2 forwarding the SIP URIs and the values to PTC 3 and PTC 4. Both PTCs will then realise a registration process with the SIP URIs to be able to receive the SIP instant messages with the temperature values by service 3. As soon as PTC 3 and PTC 4 received the messages, the values included will also be compared with the values received from PTC 2. After the test case execution is finished, an overall verdict (such as "pass", "fail" or "inconclusive") is assigned.

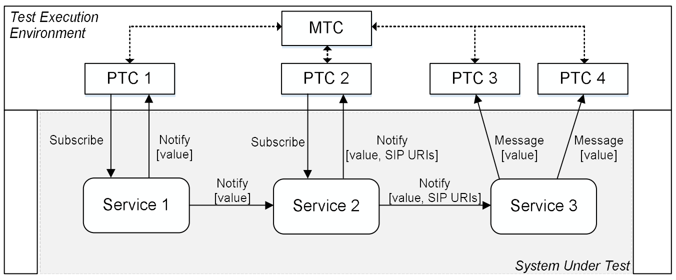

Fig. 8. Testing the M2M Application "Temperature Surveillance"

VII. Conclusion

This publication presents a novel concept for automated functional testing of P2P-based services and applications in M2M. It considers the volatility of peers providing diverse services and deals with the complexity of M2M applications which are composed of several heterogenic services. By means of TDL concepts in the approach, tester components can be identified, test data can be derived as well as reusable test behaviours for services.

In a next step, the TDL-based test behaviour will be analysed in terms of similarity and reusability. Furthermore, it needs to be analysed how tester components between services can be merged. We are simultaneously working on a concept of integrating a trust management system inside the presented testing framework for ensuring security and trustworthiness in P2P-based M2M applications using trust.

Acknowledgements

The research project P2P4M2M 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 03FH022IX5. The authors of this publication are in charge of its content.

References

[1] ETSI TR 102 725, V1.1.1, 2013-06: Technical Report, "Machine-to-Machine communications (M2M); Definitions", ETSI TISPAN

[2] M. Steinheimer, U. Trick, W. Fuhrmann and B. Ghita, "P2P-based community concept for M2M Applications," Proc. of Second International Conference on Future Generation Communication Technologies (FGCT 2013), London, UK, December 2013

[3] M. Steinheimer, U. Trick, P. Ruhrig, R. Tönjes, M. Fischer and D. Hölker, „SIP-basierte P2P-Vernetzung in einer Energie-Community", ITG-Fachbericht 242: Mobilkommunikation, pp. 64, Mai 2013

[4] M. Steinheimer, U. Trick, B. Ghita and W.Fuhrmann, "Decentralised System Architecture for autonomous and cooperative M2M Application Service Provision", International Conference on Smart Grid and Smart Cities (ICSGSC), in press, 2017

[5] E. Almeida, G. Sunye, Y. Traon and P. Valduriez, "A framework for testing peer-to-peer systems," Dans 19th International Symposium on Software Reliability Engineering (ISSRE 2008), Redmond, Seattle, USA, IEEE Computer Society, 2008

[6] A. Ulrich and H. Konig, "Architectures for testing distributed systems," Dans Proceedings of the IFIP TC6 12th International Workshop on Testing Communicating Systems, pp. 93–108, Deventer, The Netherlands. Kluwer, B.V., 1999

[7] P. Rosenkranz, M. Wählisch, E. Baccelli and L. Ortmann, "A Distributed Test System Architecture for Open-source IoT Software," IoT-Sys´15 Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems, pp. 43-48, ACM, New York. 2015

[8] C. Torens and L. Ebrecht, "RemoteTest: A Framework for Testing Distributed Systems," 2010 Fifth International Conference on Software Engineering Advances (ICSEA 2010), pp. 441-446, Nice, France, 2010

[9] A. Khoumsi, "Testing distributed real time systems using a distributed test architecture", Sixth IEEE Symposium on Computers and Communications, 2001

[10] M. Winter, T. Roßner, C. Brandes and H. Götz, "Basiswissen modellbasierter Test", dpunkt Verlag Heidelberg, Germany, ISBN: 978-3-86490-297-0, 2016

[11] D. Harel, "Statecharts: A Visual Formalism for Complex Systems," Science of Computer Programming, 8(3):231–274, June 1987

[12] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language user guide," Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1999

[13] W. Reisig, "Petri Nets: An Introduction," Springer-Verlag New York, Inc., New York, NY, USA, 1985

[14] The Unified Modeling Language 2.0: Superstructure FTF convenience document (ptc/04-10-02). Object Management Group, 2004. Available at:www.uml.org

[15] P. Wacht, U. Trick, W. Fuhrmann and B. Ghita, "Efficient Test Case Derivation from Statecharts-Based Models", Proceedings of the Eleventh International Network Conference, Frankfurt, Germany, 2016

[16] ETSI ES 203 119-1, V1.3.1, 2016-09: "Methods for Testing and Specification (MTS); The Test Description Language (TDL); Part 1: Abstract Syntax and Associated Semantics", ETSI

[17] IEEE Std 610.12, (1990), IEEE Standard, "IEEE Standard Glossary of Software Engineering Terminology", IEEE

[18] A. Saifan and J. Dingel, "Model-based testing of distributed systems," Technichal report, vol. 548, 2008

[19] S. Ghosh and A. Mathur, "Issues in testing distributed component-based systems," In Proceedings of the First International Conference on Softaware Engineering Workshop on Testing Distributed Component-Based Systems, Los Angeles, CA, May 1999

[20] M. Pezzè and M. Young, "Software testen und analysieren" (translated title: "Testing and analysing software"), Oldenbourg, Munich, Germany, ISBN: 3-486-58521-6. 2009

[21] I. Schieferdecker, "Modellbasiertes Testen," OBJEKTspektrum 3/07, pp. 39-45. 2007

[22] 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