IoT-Testware – an Eclipse project

Ina Schieferdecker
TU Berlin and
Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany
ina.schieferdecker@fokus.fraunhofer.de

Sascha Kretzschmann, Axel Rennoch,
Michael Wagner
Fraunhofer FOKUS
Kaiserin-Augusta-Allee 31
10589 Berlin, Germany

***Abstract*—It is the aim of IoT-Testware to supply a rich set of TTCN-3 test suites and test cases for IoT technologies to enable developers in setting up a comprehensive test environment of their own, if needed from the beginning of a project. Initially, IoT-Testware will focus on protocols like CoAP and MQTT. To ensure test and implementation technology independence, the test suites will be realized in TTCN-3 and implemented with Titan. TTCN-3 has been defined and standardized by the European Telecommunication Standards Institute in ETSI ES 201873 and related extension packages. It is implemented and supported in Eclipse IoT by the Titan project. The test suites will contain tests for conformance, interoperability, robustness, and security aspects.**

Keywords—IoT; Testing; Open Source; Eclipse; TTCN-3

I. INTRODUCTION

The open source community has produced a lot of excellent technology, frameworks and products that help implementing IoT applications. A developer usually selects an appropriate set of technology and components and incorporates them into an application. The chosen components need to support the implementation of all relevant aspects of an IoT solution including device connectivity, management, monitoring, and business logic and last but not least security enforcement at all levels.

Implementing test suites and test cases covering several aspects, levels, interfaces and protocols ensuring scalability, interoperability and security is a tedious task. There are currently many redundant pre-competitive activities ongoing with limited access and impact to the IoT community.

The IoT-Testware project at the Eclipse Foundation [4] set up a systematic approach for the development of automatic executable test suites for IoT relevant protocols and services. It follows the ETSI methodology for test development by using standardized notations and procedures that provides a basis for certification and labeling initiatives.

II. IOT UNDER TEST

With the emerge of the IoT, the quality assurance area face new challenges. It needs to be considered that the system under test (SUT) vary from single IoT devices to highly dynamic IoT infrastructures and platforms. Consequently, test design techniques have to be able to deal with a high number of devices with open interfaces. Those devices are sensors, actuators, microcontrollers or gateways. Sometimes they are installed in harsh and unreliable environments. Primarily, devices like sensors and microcontrollers work under resource-constrained conditions such as energy supply or network availability. All these non-functional aspects play a role and must be considered by the test environment next to the functional aspects like the software running on a device. Implementing an easy test architecture as shown in Fig. 1, a single micro controller unit (MCU) is the SUT where its functionality and its ability for communication via the Constrained Application Protocol (CoAP) [3] is tested.

The bigger the system becomes the more complex testing approaches are needed. These prove quality not only for systems themselves but also for connected system-to-system architectures. Essential components of an IoT solution is shown in Fig. 2.

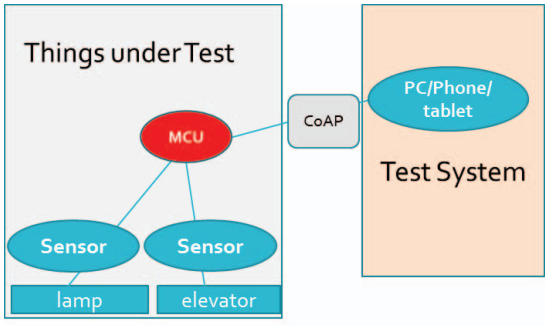

Fig. 1. Simple test architecture.

978-1-5386-0592-9/17 $31.00 © 2017 IEEE
DOI 10.1109/QRS.2017.59

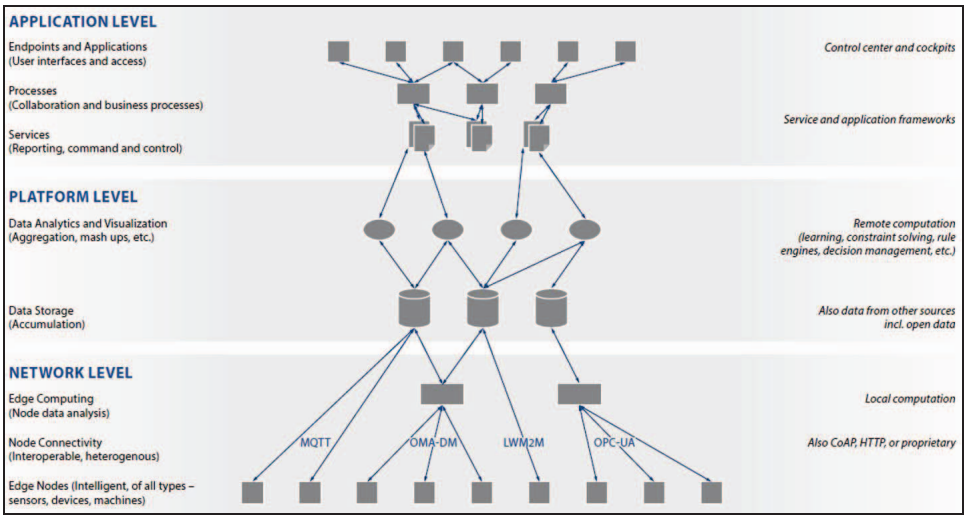

Fig. 2. IoT principal conmmunication architecture.

As indicated, even a single device can stand as discrete system assembled with other, perhaps more complex, systems. This composition is controlled and routed at the edge of the network level, normally represented by IoT-gateways. At this level, interoperability is one of the important non-functional requirement, as different system may implement different protocols. Besides interoperability, security is the main concern in the IoT development ([26][28]) as raw data and extracting insights from data can cause serious harm. Fig. 3 illustrates an IoT service test architecture that focuses on internet-accessible services using the Message Queue Telemetry Transport (MQTT) protocol [21]. The SUT is an IoT gateway or a cloud server where security, connectivity and management functionalities are main test concerns in particular.

Beyond that, integrated IoT infrastructures are tested using the IoT infrastructure test architecture as shown in Fig. 4. As SUT we consider IoT operating systems or IoT platforms like the oneM2M service layer implementations [24], the RIOT operating system [27] or any other.

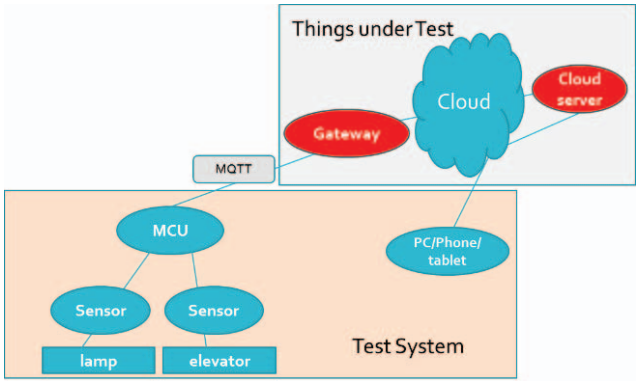

Fig. 3. Gateway test architecture.

The high number of different (IoT) protocols, each with its own advantages and disadvantages, is posing another challenge for IoT testing. Thus, conformance testing becomes important and define a significant requirement for quality management. Within the Eclipse IoT-Testware project we start to develop test suites for the de facto standard protocols in the world of the IoT, namely MQTT and CoAP.

III. The Role of TTCN-3

The Testing and Test Control Notation (TTCN-3) [16], as the only international test language standard, has been designed and defined for the abstract specification and implementation of test cases. Originally, it has been used for black-box conformance protocol testing in telecommunication only, but over the years its scope and applicability grows and today it is suitable also for performance and security testing in multiple domains like e.g. automotive, medicine, or banking.

The technology is explained very well in articles and books (see e.g. [32]). It has been accepted in several international projects, performed through e.g. European Telecommunication Standardization Institute (ETSI), e.g. SIP tests [31], 3GPP or the WiMax-Forum certification programs.

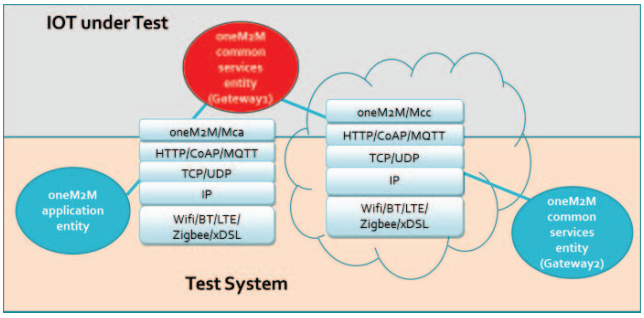

Fig. 4. oneM2M test architecture.

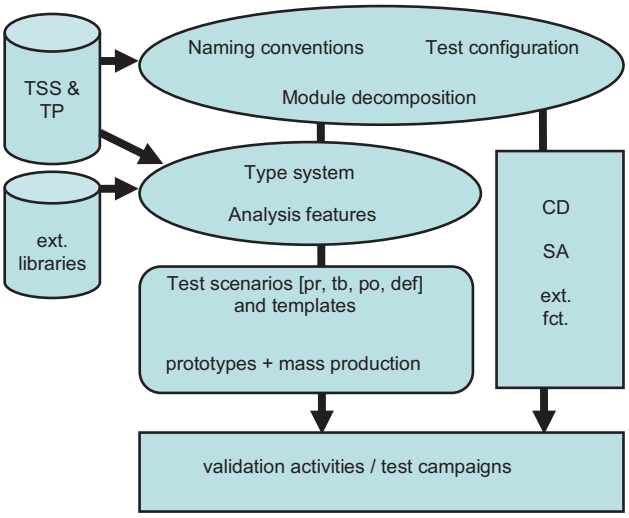

Fig. 5. TTCN-3 test development process.

As part of the ISO Conformance Testing Methodologies Framework the first version of TTCN have been defined in principle and standardized since the beginning of the nineties [19] and many test suites have been produced and executed. Many standardization bodies on international and national level, also in industry make use of TTCN-3 test suites for assuring the conformance and interoperability of their technologies. These bodies include

- GCF/3GPP LTE mobile user equipment [1],
- oneM2M (Standards for M2M and the Internet of Things) [25],
- OMA (Open Mobile Alliance Mobile Phone Standards & Specifications) [23],
- Autosar (AUTomotive Open System ARchitecture) [2],
- TETRA (Terrestrial Trunked Radio) [29],
- ETCS (European Train Control System) [6].

One of the major reasons for the success of TTCN-3 is the platform independence, ease to understand and learn the language and the tool support for the test suite development and automated execution of the test cases. A major step for the dissemination of the technology has been two years ago with the availability of the powerful Titan tool [5], an open source TTCN-3 compiler and execution environment developed by Ericsson and transferred to the Eclipse Foundation.

Today in most cases there is no formal (i.e. machine processable) system model of the SUT available and time pressure does not allow developing any formal model to get an automatic tool support for test suite derivation. Thus, a manual-written synthesis of the test suite is required. In the following, we demand the availability of an existing test plan, i.e. a test suite structure and test purposes (TSS & TP) document that provides the basis for the implementation of the abstract test suite (ATS) with TTCN-3.

In the industrial projects, the TTCN-3 test suite development process follows a clear sequence of development steps: As illustrated in Fig. 5 at the top a common set of naming conventions within the project, the finding of suitable test system interfaces and configurations for the selected test architectures and a structure for the TTCN-3 modules will be addressed. In a second step, the data type system has to be specified. Existing libraries on type definitions (TTCN-3 or other languages like ASN.1, IDL or XML) or frameworks with reusable TTCN-3 functions (e.g. [12]) have to be considered. At this point, it is also the time to agree on potential auxiliary logging features enabling e.g. the automatic production of meaningful test reports during test campaigns.

After the clarification of the above issues a prototype test scenario with preamble (pr), testbody (tb), postamble (po) and default (def) behaviour should be written and evaluated before the mass of the test cases will be implemented. System adapter (SA), codec (CD) and external functions that are part of the TTCN-3 test system architecture could be implemented in parallel to the last two steps. Please note that the Eclipse Titan project already provides an extensive set of system adapter and codec for the most relevant access ports in the context of the IoT.

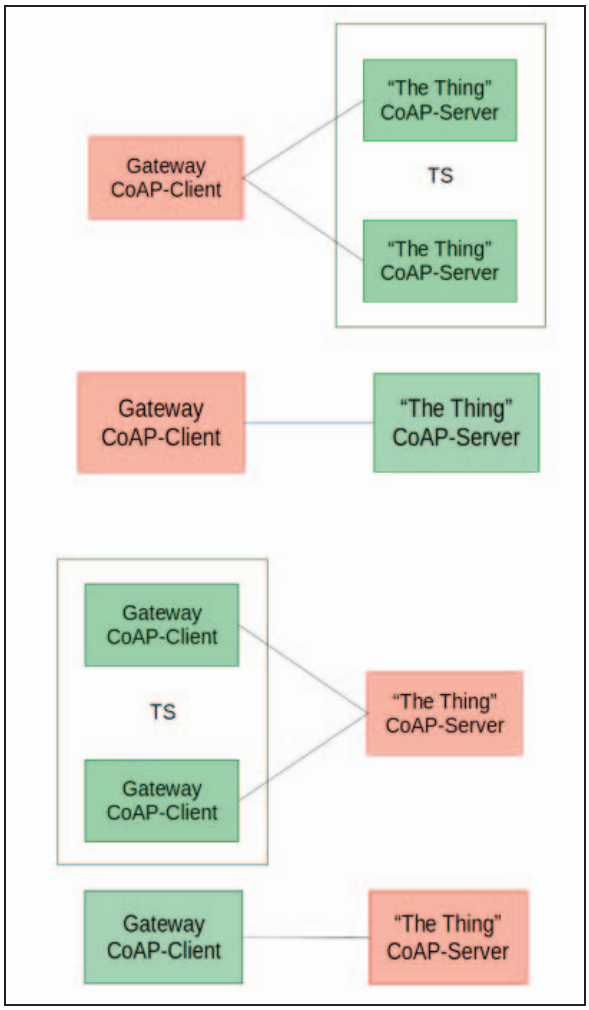

Fig. 6. CoAP test configurations

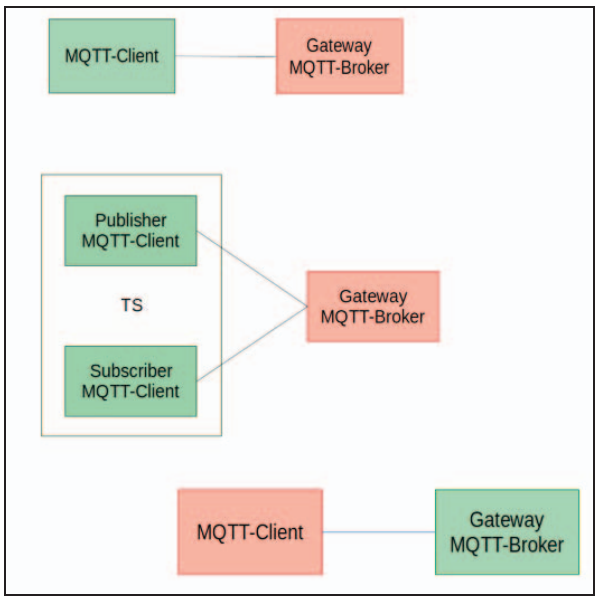

Fig. 7. MQTT test configurations

There are no standard naming conventions available and even in different ETSI projects on TTCN-3 test suites different naming conventions have been found [14][15]. The latter situation is due to individual preferences and existing naming conventions coming with the SUT standards. Interworking test suites may even apply different naming conventions for two involved protocols in order to reuse existing definitions for one of the protocols. Nevertheless, an agreement on prefixes for e.g. functions, defaults, templates are recommended [13].

The TTCN-3 test suite development process ends with validation activities that may include application of static quality checks but also sample test runs against SUT simulations or selected product implementations.

IV. Eclipse Test Suites

The Eclipse IoT-Testware project is dedicated to open source test solutions for IoT products and services as described in the previous section II. This includes the popular IoT transport layer protocols CoAP and MQTT that are part of the initial focus of the work program. Following the ETSI methodology developed and promoted by the ETSI Centre for Testing and Interoperability [7], the work started with the identification of the relevant test architecture and the development of the TSS and TP catalogues. Fig. 6 and Fig. 7 illustrate the most relevant test configurations for CoAP and MQTT. The initial test suites structures include basic checks of mandatory message data fields but also dynamic tests of protocol features for both server and client roles of the SUT. The initial scopes of both TSS are given in the Fig. 8 and Fig. 9.

The functional test case definitions will also be extended by performance and security tests. For security testing it is intended to make use of the open source Fuzzino library [17] to allow the generation of test data for fuzz testing.

CoAP Server as SUT
- All mandatory message data fields
  - Support all defined method codes and understand regular and illegal or corrupted data along with them
- Protocol features
  - General
    - Block transfer
    - Piggybacked responses
    - Message Types (ACK, CON, NON-con, ReSeT)
  - Options
    - Max-Age
    - Token option
    - Several URI-path options
    - Several URI-query options
  - Lossy context
  - Discovery service
  - Error handling

Fig. 8. CoAP test suite sturcture

Both CoAP and MQTT test purpose definitions are following the usual TP formats as used in ETSI projects, see samples in Fig. 10 and Fig. 11.

MQTT Broker as SUT
- All mandatory message data fields
  - Regular and illegal/corrupted data
    - Fixed Header
    - Variable Header
    - Payload
      - Client identifier length restriction (up to 65535 bytes)
      - UTF-8 encoding
- Protocol features
  - General
    - QoS levels
    - Delivery retransmission
    - Retained messages
    - Message ordering
    - Anonymous client identifier
  - Connect/disconnect (session handling)
    - Credentials
    - Session initiation
    - Session states
  - Subscribe
  - Unsubscribe
  - Immediate publish (w/o awaiting for CONNACK)
  - Last Will and Testament (LWT) message
  - Heartbeats: keepAlive values (max timeout between message exchange)
  - Topic names/filters
  - Error handling

MQTT Client as SUT
- All mandatory message data fields
  - Regular and illegal/corrupted data
  - UTF-8 encoded Strings

Fig. 9. MQTT test suite sturcture

TP-ID	TP_CoAP_Server_001
Selection	PIC_Server
Summary	The IUT is responding on a RESET message.
Reference	RFC7252#section4.2
Initial condition	none
Test purpose	
Ensure that the IUT on receipt of an EMPTY message containing msg_type := 0 (CONfirmable) containing code := 0.00 sends a RESPONSE message containing msg_type := 3 (ReSeT)	
Comments	

Fig. 10. Test purpose sample for CoAP

TP-ID	TP_MQTT_Broker_CONNECT_001
Selection	PIC_Broker
Summary	The IUT MUST close the network connection if fixed header flags in CONNECT Control Packet are invalid
Reference	[MQTT-2.2.2-1], [MQTT-2.2.2-2], [MQTT-3.1.4-1], [MQTT-3.2.2-6]
Initial condition	
Test purpose	
Ensure that the IUT on receipt of an CONNECT message containing header_flags := '1111'B sends no RESPONSE message and closes the Network Connection	
Comments	

Fig. 11. Test purpose sample for MQTT

V. Benefits for the Community

Providing an open test suite to the vendors of IoT systems and solutions is one possible way to meet the required quality of such implementations. ETSI for example, provides standardized publicly available test suites (conformance, interoperability, performance benchmarking etc.) implemented with TTCN-3 for almost every of its standardized communication protocol or service specification across domain borders. This list [11] encompasses test suites for the

- Tunneling Protocol (GTP) for the General Packet Radio Service (3GPP GPRS),
- WiMax (IEEE 802.16),
- ePassport Readers,
- Session Initiation Protocol (IETF SIP),
- IP Multimedia Subsystem (3GPP and ETSI INT),
- DIAMETER (an IETF authentication, authorization, and accounting protocol),
- IPv6 (IETF),
- Digital Private Mobile Radio (ETSI dPMR),
- Digital Mobile Radio (ETSI DMR),
- Intelligent Transport Systems (ETSI ITS) Test Suites.

Vendors or implementers of products according to these standards do have immediately an executable test suite at hand with which they can validate whether their implementation can claim conformance with the specification. This has several advantages for the vendors, which are among others:

- Early availability of a validated and consolidated test suite for (parts of) the requirements,
- more time for the development of the system,
- early feedback regarding compliance of the developed system with the respective standard,
- reduced costs for the development of the conformance test suite,
- clarification of ambiguous requirements specifications with respect to room for interpretations,
- unbiased evaluation of the implementation's conformance, and
- independent triage of test cases for the conformance test suite.

There are also opinions who see some risks in publishing open test suites. In the following, we like to address two major issues.

The availability of a publicly available test suite could tempt the industry to adjust their implementation to what is required for passing the tests, instead of implementing the requirement specifications. As a result, the implementation would cover most probably only the functionality (and, thus, requirements) covered by the standardized test suite. On the other side if the test suite is able to determine that the implementation meets the at least required quality with respect of covered requirements, the passing of all the tests would give a great deal of confidence.

The manipulation of test results in order to get the approval is a well-known problem in technical engineering disciplines, at the latest since e.g. the diesel exhaust scandal. It means that the implementation/subject of testing adjust its behavior to what is expected from the test environment. An attempt of defraud by manipulations of the products could never be

excluded. However, such attempts do not have any meaning in the context of the idea for an open test specification and test system. The risk of faked test results may be much higher if vendors claim "passed" test verdicts in their test report

Beside such discussion we see many more multiple benefits for the IoT and Eclipse communities due to the IoT-Testware project, e.g. a common growing test suite pool provided with one unique international accepted test language that is ready for download and use in your environment. The tests are open source and under the professional control of the Eclipse community and can be extended following the future demands of the technology and market requirements.

VI. FUTURE WORK: TTCN-3 VIRTUALIZED

As stated in the previous chapters, we gain many advantages when using open-source software within the IoT testing domain. Thus, we can benefit from taking the best from the tools to increase the degree of automation. We have learned from our first steps in the IoT testing field that a big pitfall is still the set-up of a full test (automation) environment, including test suites and associated tools as well as third party tools for tasks like monitoring, logging or to run security tests like fuzzing. The composition of all these available open source tools is a non-trivial task but needs to be tackled as it reflects the complexity of IoT infrastructures also for the test domain. A logical consequence is to hide the complexity from the vendors or implementers of products who want to ensure quality with the help of the test suites. We suggest two advanced IoT testing approaches named "TTCN-3 virtualized" and "virtualized testing with TTCN-3".

Virtualizing the TTCN-3 test suites is a straightforward solution and has the goal to deploy the Eclipse IoT-testware together with the Titan ecosystem as a service. This service provides the possibility to either use existing test suites as defined in the Eclipse IoT-testware project or self-written test suites that follow the official TTCN-3 language standard. In any case, the Titan ecosystem comes with the needed test ports (adapters and codecs to allow the communication with the SUT) and the Titan compiler that compiles runnable test code out of the TTCN-3 modules that can be executed against your SUT. Logging is not restricted to be done natively by Titan, as users are free to implement their own logging architecture on the target or using a third-party tool. Besides some configurations like choosing the right set of test suite or providing configurations of the SUT, virtualized TTCN-3 hides the complexity of the test environment. However, it needs to be consider that tests might not running on highly constrained devices implementing this approach. Mostly, these devises are not designed to reply to incoming stimuli (e.g. simple MCUs sending sensor data). Furthermore, virtualized TTCN-3 is limited to few test requirements like functional and conformance testing. Configurations that are more complex could need too much manual work and would be less advantageous.

The virtualized testing approach is another idea and similar to the TTCN-3 virtualized approach but it defines another main goal. Unlike foregrounding the building and executing conformance tests as defined the Eclipse IoT testware, the user is provided with a fully integrated test ecosystem as drafted in Fig. 12.

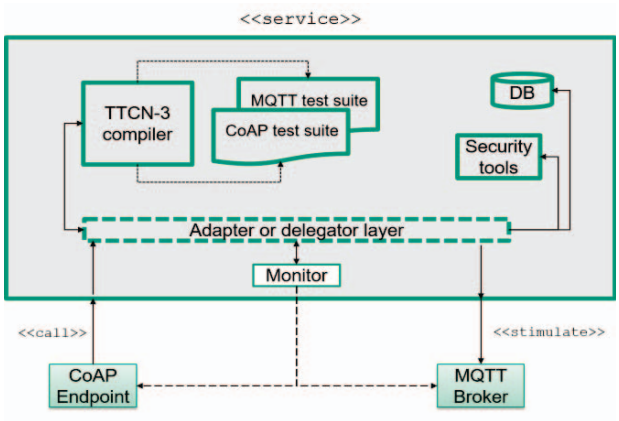

Fig. 12. Virtualized testing with TTCN-3

We suggest a service that can be used by vendors. In their perspective the virtualized test system appears either as API they call with the SUT or they configure a kind of website to let the service stimulate the SUT. It is shown in Fig. 12 that a CoAP endpoint (e.g. CoAP client) calls the service to get tested whereas a MQTT broker must be stimulated as it doesn't play an initializing role within the MQTT protocol. Inside the "black-box" the requests from or the stimulations against the external test objects will be processed. An adapter undertake the task of delegating requests. In addition, it includes loading the appropriate configuration and initiates all the involved tools (e.g. third-party tools, databases). As said for virtualized TTCN-3, the Titan ecosystem is running as service and provides support for compiling and executing test cases. Based on the parameter transmitted by the adapter layer, Titan executes the correct test suite together with the fitting ports. In the example illustrated in Fig. 12 the embedded Titan compiler loads the CoAP test suite if the service called by a CoAP endpoint or provides the adapter layer with the MQTT test suite to stimulate the MQTT broker, respectively. We recommend designing the virtualized testing service as plugin architecture to easily add functionality later like third-party tools, test suites or adapters.

Furthermore, we see a potential in extending the Node-RED tool by a testing portion: With Node-RED it is possible to simply build its own IoT infrastructure by adding nodes (e.g. devices, APIs or online services) graphically and connect them via edges [22]. We illustrated a toy example in Fig. 14, where a local CoAP server is tested using Node-RED. The most left blue node triggers two outgoing flows that lead to a function node (orange) setting up a global counter. The other flow ends in a representation of a CoAP server running locally and sending GET requests to an observable resource named "Observable_Resource" resulting in an outgoing flow to the "Test component" node. Configurations are part of the tool and shown in Fig. 13 for the local server. The "Test component" node is a function node and can be seen as test case as it holds the test logic. In the provided example it simply checks if the global counter is equal to the value of the "Observable_Resource" (both, the global counter and the

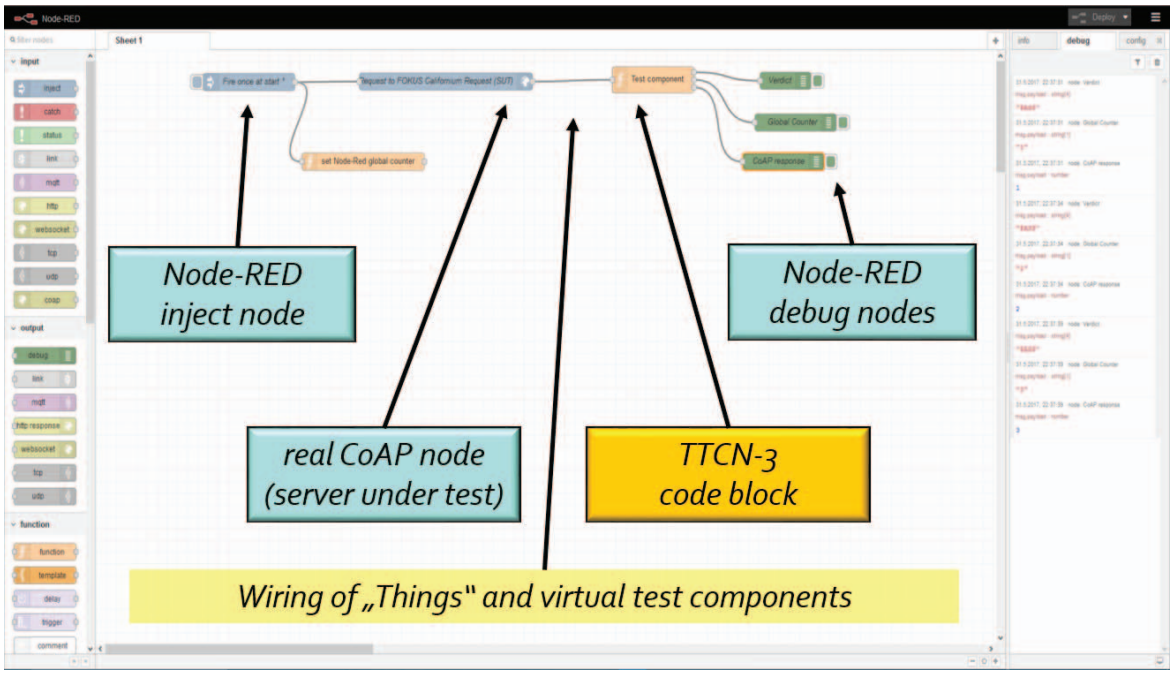

Fig. 13. Node-RED included TTCN-3

resource value are incremented, after every second and in the "Test component" node, respectively). Lastly, the result of the test node ends in three different debug nodes (green) that are used to log the execution because it logs the verdict, the current value of the global counter and the current value of the "Observable_Resource". The logging is displayed in Fig. 13, on the right "debug tab. This is one possibility to introduce testing into Node-RED.

VII. Conclusion

Following the methodology and the discussion of the technical approach presented in this paper the authors see very good chances and results on the open source approach using a mature and standardized methodology. It is of great benefit for the industry to refer to a commonly developed and confirmed test suite using free testing tools. Furthermore, the approach based on TTCN-3 provides additional opportunities for standardization and certification bodies if more than a critical mass is applying and accepting the results. Finally, with the virtualized approach, at which virtualized testing can be seen as an extended approach of virtualized TTCN-3, it becomes possible to open the technology to everyone who needs to ensure quality for ones IoT product independent of the complexity to a certain extent.

Fig. 14. Node-RED in-build CoAP server configuration

Acknowledgment

The authors appreciate the Eclipse foundation, in particular the members of the IoT-Testware and Titan projects for their contributions. Especially we thank Mr. Alexander Kaiser of the relayr GmbH for his contribution on the MQTT test suite in the IoT-T project [20] and the support of the Open source approach.

References

[1] 3GPP Test suites: ftp://ftp.3gpp.org/tsg_ran/WG5_Test_ex-T1/TTCN/Deliveries/

[2] Autosar: BSW & RTE Conformance Test Specification, https://www.autosar.org/fileadmin/files/releases/4-0/conformance-testing/AUTOSAR_PD_BSWCTSpecCreationValidation.pdf

[3] Constrained Application Protocol: http://coap.technology/

[4] Eclipse IoT-testware: https://projects.eclipse.org/projects/technology.iottestware

[5] Eclipse Titan: https://projects.eclipse.org/projects/tools.titan

[6] ERTMS (The European Rail Traffic Management System), http://www.ertms.net/

[7] ETSI Centre for Testing and Interoperability: http://www.etsi.org/about/how-we-work/testing-and-interoperability/centre-for-testing-and-interoperability

[8] ETSI Conformance Testing for the Network layer of HiperMAN/WiMAX terminal devices; Part 3: Abstract Test Suite (ATS) http://www.etsi.org/deliver/etsi_ts/102600_102699/10262403/01.02.01_60/ts_10262403v010201p.pdf

[9] ETSI Making Better Standards, https://docbox.etsi.org/MTS/MTS/10-PromotionalMaterial/MBS-20111118/home.htm

[10] ETSI Plugtests, http://www.etsi.org/about/what-we-do/plugtests

[11] ETSI Test suites: http://www.ttcn-3.org/index.php/downloads/publicts/publicts-etsi

[12] ETSI TTCN-3 libraries: http://www.ttcn-3.org/index.php/development/devlibraries

[13] ETSI generic naming conventions http://www.ttcn-3.org/index.php/development/naming-convention

[14] ESTI TS 102 027-2: TIPHON Technology Compliance Specification, IETF SIP 3261, ATS and partial PIXIT. ETSI 2003.

[15] ETSI TS 102 351: TTCN-3 IPv6 Test Specification Toolkit, ETSI 2004.

[16] ETSI European Standard (ES) 201 873 The Testing and Test Control Notation version 3; http://www.ttcn-3.org

[17] Fraunhofer FOKUS, Fuzzino library: https://github.com/fraunhoferfokus/Fuzzino

[18] Global Certification Forum: Globally Trusted Devices for Digital Life. http://www.globalcertificationforum.org/images/gallery/Events/2015_Bengaluru_Workshop/Bengaluru_Workshop_5_April_2016_part1.pdf

[19] ISO 9646 / ITU-T: Conformance Testing Methodology Framework, multipart standard.

[20] IoT-T project: http://www.iot-t.de/en/

[21] Message Queue Telemetry Transport: http://mqtt.org/

[22] Node-RED: https://nodered.org

[23] OMA: Interoperability Working Group, http://technical.openmobilealliance.org/Technical/technical-information/working-groups-and-committees/interoperability-working-group

[24] oneM2M Functional architecture: http://onem2m.org/images/files/deliverables/Release2/TS-0001-%20Functional_Architecture-V2_10_0.pdf

[25] oneM2M Test suite (in progress): http://git.onem2m.org:8080/TST/ATS/tree/master

[26] G. Press: Internet Of Things By The Numbers: What New Surveys Found; https://www.forbes.com/sites/gilpress/2016/09/02/internet-of-things-by-the-numbers-what-new-surveys-found

[27] RIOT Real time OS for sensor networks: https://riot-os.org/

[28] I. Skerrett: IoT Developer Trends 2017 Edition; https://ianskerrett.wordpress.com/2017/04/19/iot-developer-trends-2017-edition/

[29] TCCA (TETRA and Critical Communications Association), http://www.tandcca.com/

[30] WiMAX Forum Certification Programs http://www.wimaxforum.org/Page/certification

[31] A. Wiles et al.: Experiences of using TTCN-3 for Testing SIP and OSP. ETSI, Sophia-Antipolis 2001.

[32] C. Willcock et al.: An Introduction to TTCN-3, 2nd Edition. Wiley, 2011.