R2.2: Testarchitekturen

Gefördert durch:

Bundesministerium für Wirtschaft und Energie

aufgrund eines Beschlusses des Deutschen Bundestages

Version 1.0, Datum: 30.06.2017

Autoren:

Frank-Walter Jäkel	- Fraunhofer IPK
Stefan Stoelzle	- AUDI AG
Paul Hopton	- Relayr
Yuliya Brynzak	- Relayr
Alexander Kaiser	- Relayr
Sascha Kretzschmann	- Fraunhofer FOKUS
Michael Wagner	- Fraunhofer FOKUS
Axel Rennoch (Ed)	- Fraunhofer FOKUS
Rutten, Stefan	- DEKRA
Andre Wardaschka	- DEKRA

Inhalt

1. Einleitung 3
2. TTCN-3-Testsystemarchitektur 3
3. Architektur ausgewählter Werkzeuge 3
3.1. Eclipse Titan 4
4. IoT-Testarchitekturen 5
5. Testsuiten der Konformitätstests 7
5.1. CoAP 7
5.2. MQTT 8
5.3. OPC-UA 10
6. Zusammenfassung und Ausblick 10
7. Referenzen 11

1. Einleitung

Im vorliegenden Dokument soll ein erster Überblick der Testarchitekturen und -konfigurationen erläutert werden. Das Dokument wird im Projektverlauf fortlaufend aktualisiert um weitere Testwerkzeuge in die Testarchitektur einzubeziehen.

2. TTCN-3-Testsystemarchitektur

Bei der konkreten Implementierung von TTCN-3 setzen die Entwicklungswerkzeuge auf die Referenzimplementierungsarchitektur sowie deren zwei standardisierten Schnittstellen TRI (TTCN-3 Runtime Interface) und TCI (TTCN-3 Control Interface) auf.

Diese Schnittstellenbeschreibungen zielen darauf ab, die einfache Anpassbarkeit einer TTCN-3 Executable (TE) Testsuite an eine gegebene Testinfrastruktur zu gewährleisten. Dabei passt TRI die Kommunikationsschnittstellen des Systems unter Test an das Testsystem an. Im System-Adapter (SA) werden die eigentliche Umsetzung der TTCN-3 Funktionalität wie call (bei synchroner Kommunikation) oder send (bei asynchroner Kommunikation) implementiert. Im Plattform-Adapter (PA) werden Operationen zur Implementierung von Timer-Operationen und für die Realisierung von externer (außerhalb von TTCN-3 liegender) Funktionalität zur Verfügung gestellt. TCI hingegen verwaltet die Ausführung der TTCN-3-Testsuite (Test Management / TM), realisiert die Verteilung der Testkomponenten (Component Handling / CH) und übernimmt die Testlaufprotokollierungsowie das Kodieren und Dekodieren der Testdaten (Codec / CD) (siehe Abbildung).

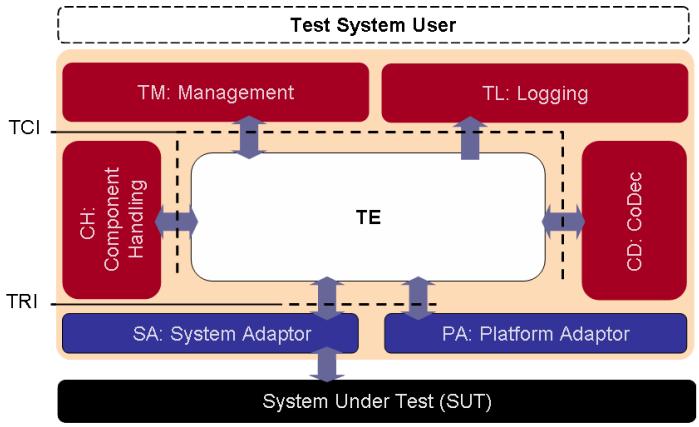


Abbildung 1: Die TTCN-3 Referenzimplementierungsarchitektur.

3. Architektur ausgewählter Werkzeuge

In diesem Abschnitt sollen die Testkonfigurationen sowie die in AP2.1 genannten Werkzeuge schrittweise gesammelt und erläutert werden, welche Werkzeuge die zu definierenden Tests der

jeweiligen Konfigurationen ermöglichen bzw. unterstützen. Das verbindende Testwerkzeug ist das TTCN-3 Open source Tool Titan. Es ermöglicht die spätere Einbindung und automatisierte Steuerung weiterer einzelner Testwerkzeuge.

3.1. Eclipse Titan

Eclipse Titan [4] ist eine Open-Source Integrations- und Ausführungsumgebung für in TTCN-3 geschrieben Testfälle. Entwickelt wird Titan bei Ericsson und bietet neben den Kernkomponenten eine grafischen IDE auf Basis von Eclipse. Zu den Kernkomponenten gehören der TTCN-3/ASN.1 Compiler, welcher aus TTCN-3/ASN.1-Modulen C++ compiliert. Dieser ist ausführbar und wird gegen das zu testende System ausgeführt. Des Weiteren unterstützt eine Basis-Bibliothek die Generierung des ausführbaren Testcodes. Diese ist in C++ geschrieben und beinhaltet wichtige Hilfsfunktionen. Schließlich sollten an dieser Stelle auch die Testports genannt werden, welche die Kommunikation zwischen dem Testsystem und dem SUT vereinfachen. Bis auf protokollspezifische Testports sind die genannten Komponenten Teil des Titan Kerns [5].

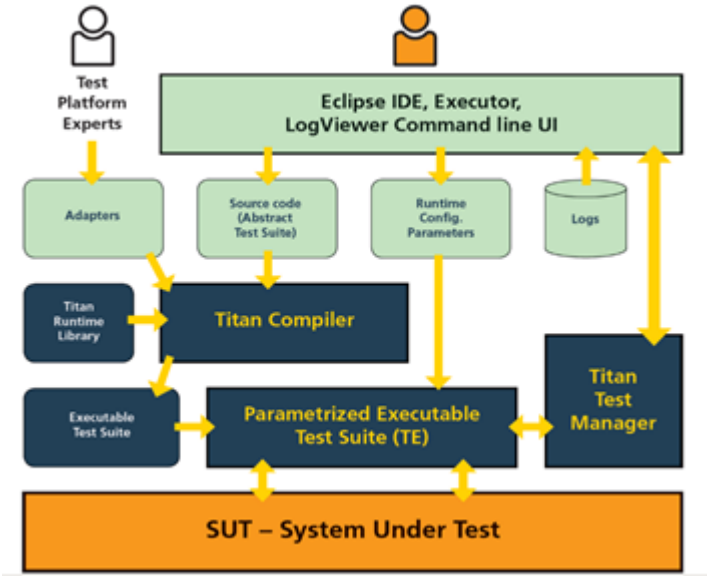


Abbildung 2: Die Eclipse Titan Architektur.

Der aus den Testmodulen generierte Code ist Protokoll-unabhängig, so dass für verschiedene Protokolle auch verschiedene Testports implementiert werden müssen. Diese Aufgabe übernimmt idealerweise ein Experte (Testplattform-Experte). So kann das Testsystem über die Testports das SUT stimulieren und Antworten empfangen. Neben dieser Adapterschicht enthalten Testports auch die Vereinfachte Erstellung von Codecs, so dass Datentypen des zu testenden Systems auf die TTCN-3/ASN.1-Typen abgebildet werden können.

Eclipse Titan dient als zentrales Werkzeug zum IoT-Testen und soll, neben der Generierung von ausführbarem Testcode, um externe Werkzeuge erweitert werden. Für das Projekt stehen von Ericsson

entwickelte Testports für die Protokolle MQTT und CoAP zur Verfügung und werden auch bereits in den jeweiligen Testsuiten verwendet (siehe Abschnitt 5).

4. IoT-Testarchitekturen

Die wesentlichen Komponenten einer IoT-Lösung und deren Vernetzung ist in Abbildung 3 dargestellt. Diese grobe Vernetzungsarchitektur nutzt Anleihen der IoT-Architekturvorschläge des europäischen F&E-Projekts IoT-A [9], von CISCO [10] und Eclipse [11].

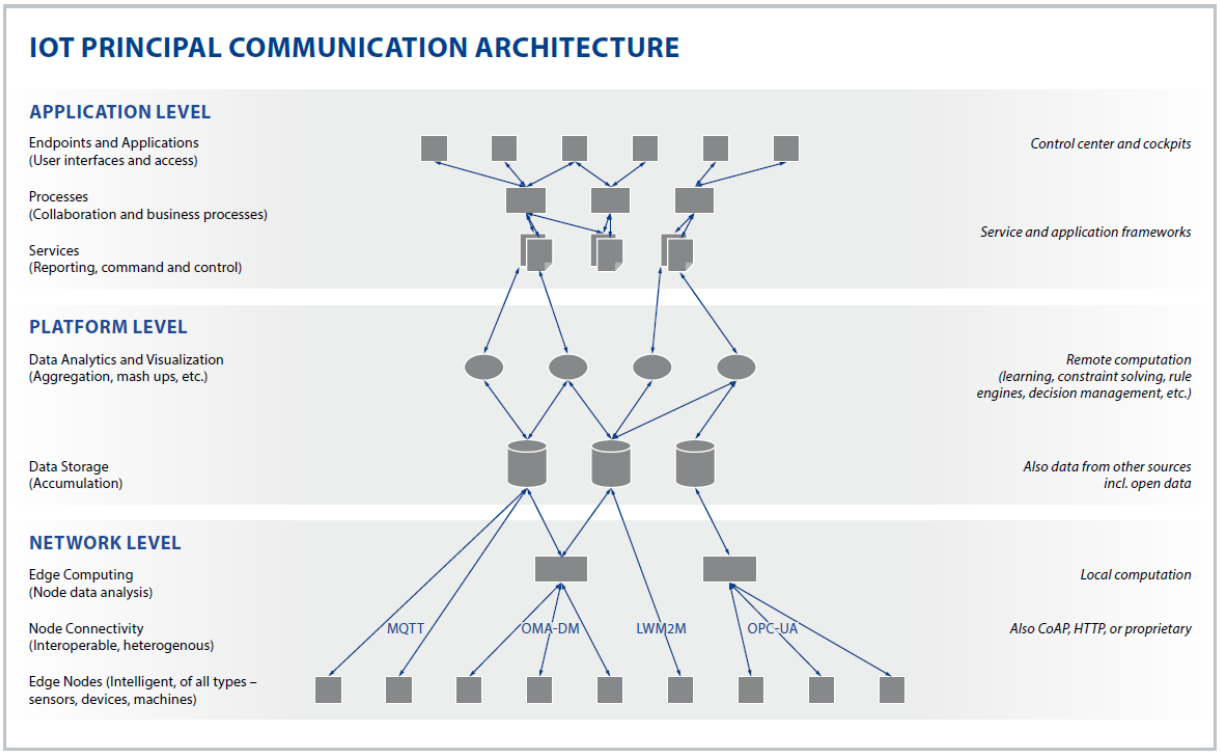


Abbildung 3: Grobarchitektur der Kommunikationswege in IoT-Lösungen der IoT-QE Arbeitsgruppe.

Aus der IoT Principle Communication Architecture lassen sich verschiedene Testkonfigurationen herleiten (s.a. [6]).

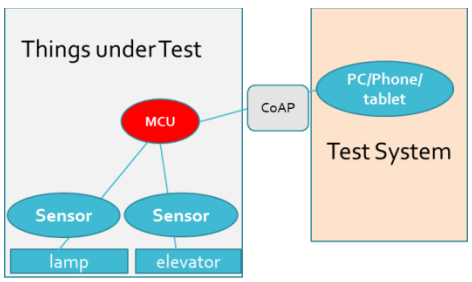


Abbildung 4: IoT-Sensorkit-Testarchitektur.

Bei der einfachen IoT-Testarchitektur (siehe Abbildung 4) ist ein Sensorkit – eine MCU (Micro Control Unit) – das Testobjekt. Das Sensorkit wird dabei auf seine Funktions- und Kommunikationsfähigkeiten beispielsweise über CoAP, MQTT, oder LwM2M untersucht.

Die über das Internet erreichbaren IoT-Dienste stehen im Fokus der IoT-Dienst-Testarchitektur in Abbildung 5. Dabei sind IoT-Gateways wie das Bosch XDK [12] oder die relayr Plattform [13] die Testobjekte. Sie werden insbesondere auf Vernetzungs-, Sicherheits- und Managementfunktionen untersucht.

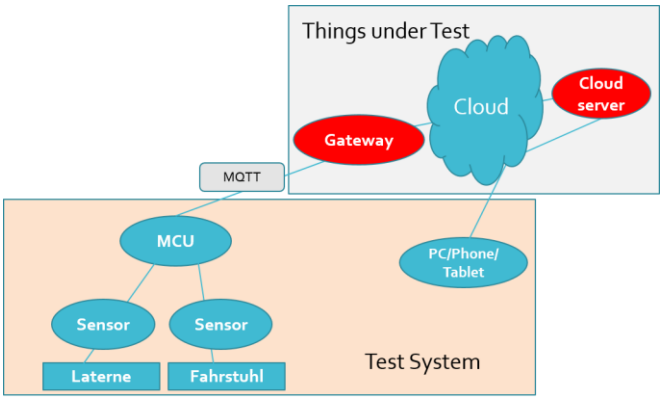


Abbildung 5: IoT-Dienst-Testarchitektur.

Ganze IoT-Infrastrukturen werden mit der IoT-Infrastruktur-Testarchitektur in Abbildung 6 überprüft. Dabei sind die IoT-Betriebssysteme bzw. IoT-Plattformen wie die oneM2M Service Layer Implementations [15], das RIOT Operating System [14] oder andere Plattform-Komponenten [11] die betrachteten Testobjekte.

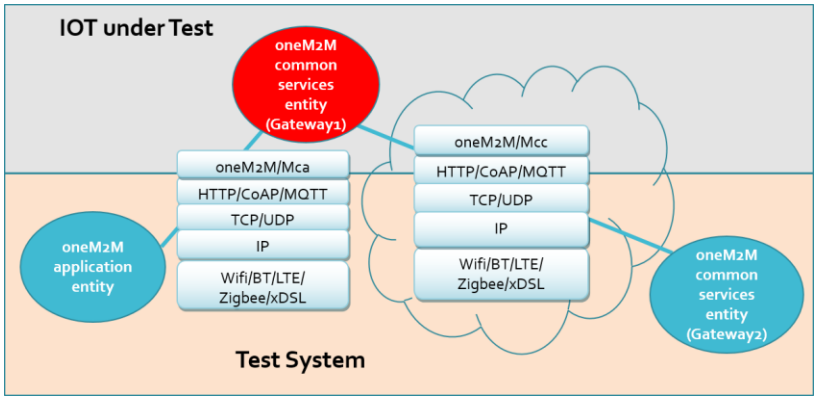


Abbildung 6: IoT-Infrastruktur-Testarchitektur.

Die Funktionalität, Skalierbarkeit und Sicherheit der auf IoT aufsetzenden Geschäftsprozesse kann Elemente aller drei genannten Testarchitekturen nutzen, um Zusatzinformationen auf Sensor/Aktuator-, Gateway- und/oder Plattform-Ebene zu nutzen.

5. Testsuiten der Konformitätstests

Der initialen Testsuiten beinhalten Konformitätstests für CoAP und MQTT, deren Testkonfigurationen in den beiden nächsten Abschnitten vorgestellt werden.

5.1. CoAP

Im vorliegenden Dokument wird das CoAP-Protokoll in der Version RFC7252 [7] verwendet. Die Terminologie wird daher aus der genannten Version verwendet, um die Konsistenz zu wahren.

Im CoAP-Protokoll teilnehmende Entitäten werden als Endpunkte bezeichnet, die auf einem Knoten „leben“. Dabei agieren Endpunkte als Sender (Quelle) und Empfänger (Ziel). In Abbildung 7 sind schematisch die Rollen in einer CoAP-Architektur dargestellt.

Ein CoAP-Server ist allgemein ein Endpunkt, der Ziel einer Anfrage und der Ursprung einer Antwort ist. Zumeist ist ein CoAP-Server mit Dingen verbunden oder einer eingeschränkten Umgebung (eine Menge von Dingen).

Ein CoAP-Client ist allgemein ein Endpunkt, der Ursprung einer Anfrage und das Ziel einer Antwort ist.

Ein Proxy hält die Rolle eines Servers und eines Clients inne. Er dient als Mittelsmann; beispielsweise zwischen verschiedenen Netzwerk, in denen unterschiedliche Protokolle für die Kommunikation eingesetzt werden.

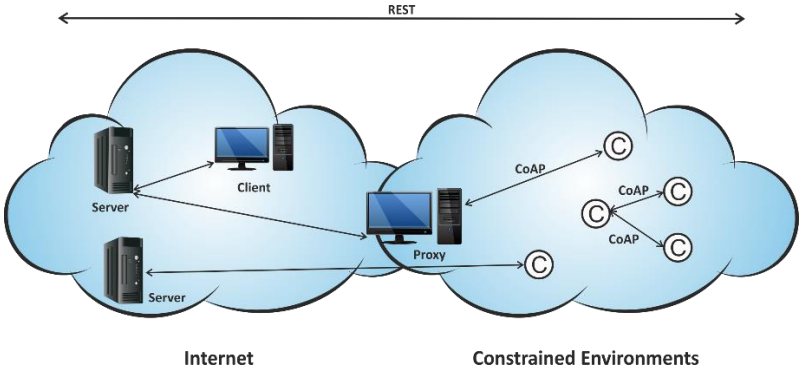


Abbildung 7: Schematische CoAP-Architektur mit den Rollen Proxy, Server und Client.

Die für die Testsuite zugrunde gelegten Testkonfigurationen sind in der folgenden Abbildung 8 und Abbildung 9 wiedergegeben.

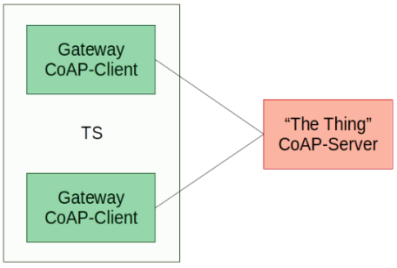


Abbildung 8: CoAP Server in der Role des SUT.

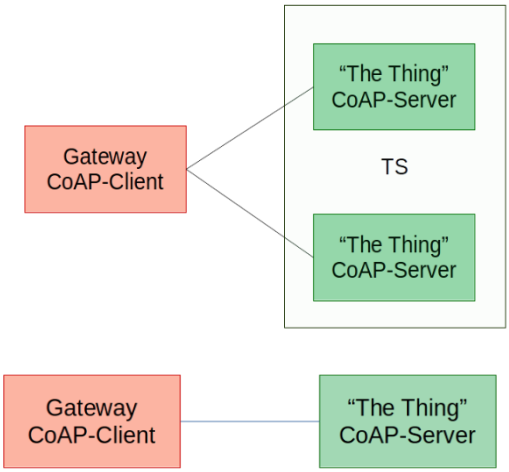


Abbildung 9: CoAP Client in der Role des SUT

5.2. MQTT

Im vorliegenden Dokument wird das MQTT-Protokoll in der Version MQTT Version 3.1.1 Plus Errata 01 [8] verwendet. Die Terminologie wird daher aus der genannten Version verwendet, um die Konsistenz zu wahren.

MQTT ist eine Client Server publish/subscribe Datentransfer Protokoll, das auf Applikationsebene (nicht zwangsläufig physisch oder auf der Ebene des Netzwerkes) eine sternförmige Netztopologie besitzt. Die Übertragung einer Nachricht wird über einen Broker (1) in zwei separate Schritte geteilt. Jeder dieser beiden Teilschritte erfolgt strikt nach dem Client Server Model, s. Abbildung 10. Die Kommunikation zwischen einem Sender (Publisher (2)) und beliebig vielen Empfängern (Subscriber (3)) ist im MQTT Protokoll über das publish/subscribe Modell realisiert.

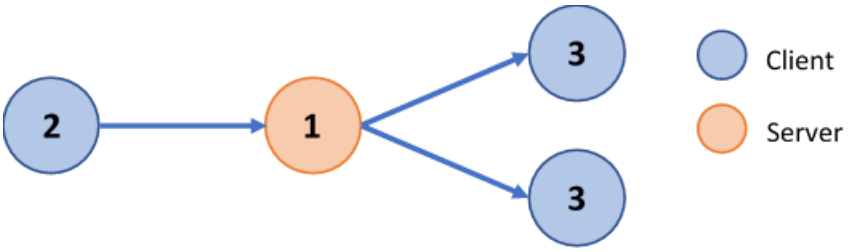


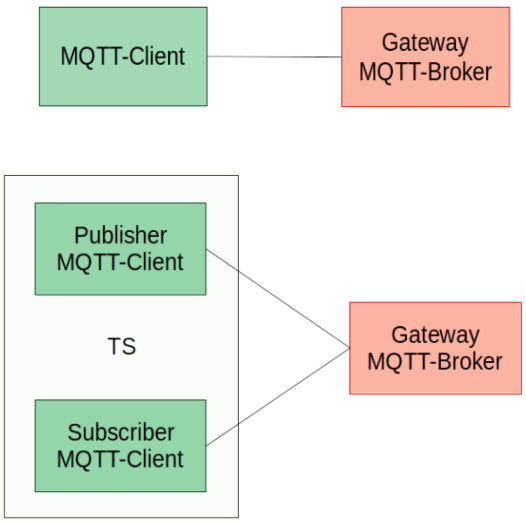
Abbildung 10: MQTT Client Server Model.

Ein MQTT-Broker ist das zentrale Bindeglied zwischen allen beteiligten MQTT-Clients. Es obliegt der Verantwortung des Brokers sämtliche Verbindungen zu den Clients und deren Subscriptions zu verwalten.

Als Publisher wird im MQTT Kontext ein Client bezeichnet der eine Nachricht an ein Topic sendet. Ein Client der eine Nachricht „publishen“ will, sendet diese nur an den Broker und an ein bestimmtes Topic. Da der Publisher keine Kenntnis über andere Clients besitzt, hat er keine Möglichkeit zu wissen wie viele – und ob überhaupt – andere Clients seine Nachricht empfangen werden.

Als Subscriber wird im MQTT Kontext ein Client bezeichnet der über eine „Subscription“ Interesse an einem bestimmten Topic angemeldet hat.

Die für die MQTT Testsuite zugrunde gelegten Testkonfigurationen sind in der folgenden Abbildung 11 und Abbildung 12 wiedergegeben.



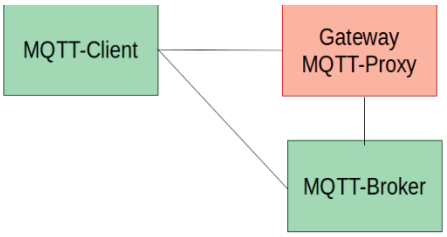


Abbildung 11: MQTT Server/Gateway in der Rolle des SUT.

Abbildung 12: MQTT Client in der Rolle des SUT.

5.3. OPC-UA

OPC-UA ist ein in der Industrie im Zusammenhang mit IoT ein häufig vorgefundenes Framework und Protokoll. Es beinhaltet Architekturaspekte (Clients und Server) aber auch Protokollaspekte wie Mechanismen zur Sicherheit und Interoperabilität. Ein Teil der in WP1 erfassten Prüfanforderungen bezieht sich auf OPC-UA, wenngleich die Prüfanforderungen im wesentlichen Protokoll und Framework übergreifend sind.

Ein erster Abgleich der Prüfanforderungen bezogen auf OPC-UA insbesondere auf der Basis eines Berichtes des Bundesamtes für Sicherheit vom 25.04.216 „Sicherheitsanalyse OPC-UA“. Der Bericht und der Abgleich mit den Prüfanforderungen zeigt, das UPC-UA hohen Sicherheitsanforderungen genügen kann aber dieses stark Implementierungsabhängig ist. Daher sind Tests erforderlich um den Grad der Umsetzung der Anforderungen an Sicherheit und Interoperabilität in der jeweiligen Realisierung möglichst automatisch prüfen zu können. Das soll im IoT-T Projekt exemplarisch gezeigt werden.

6. Zusammenfassung und Ausblick

In diesem Bericht wurden die Architekturen für IoT-Testware aus verschiedenen technischen Richtungen betrachtet. Hierzu gehören die offene standardisierte TTCN-3 Testsystemarchitektur sowie die Realisierung im Eclipse Titan Werkzeug, das die zentrale Rolle zur automatisierten Steuerung weiteren spezialisierter Testwerkzeuge übernimmt. Weiterhin sind typische Testarchitekturen im IoT-Kontext vorgestellt. Am Beispiel der für die initiale IoT-Testware ausgewählten Testsuiten für CoAP und MQTT wurden die konkreten Testkonfigurationen vorgestellt, die bei der Formulierung der Testzielkataloge und TTCN-3 Implementierungen Verwendung finden. Dieses Dokument wird schrittweise erweitert durch die Einbindung ausgewählter Testwerkzeuge.

7. Referenzen

[1] T. Vassiliou-Gioles, I. Schieferdecker: Testen mit TTCN-3 - Vorteile und Nutzen einer mächtigen und zudem standardisierten Testtechnologie. In telekom praxis, tp 11/12-2007.

[2] ETSI The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)

[3] ETSI The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI)

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

[5] User Guide for TITAN TTCN-3 Test Executor, Seite 4 f.

[6] IoT-T Projektbericht R2.1: Stand der Testtechniken im Projekt, Juni 2017

[7] IETF RFC7252: The Constrained Application Protocol (CoAP), https://tools.ietf.org/html/rfc7252

[8] MQTT Version 3.1.1 Plus Errata 01, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[9] EU FP7 Projekt Internet of Things Architecture, Updated reference model for IoT, siehe http://www.meet-iot.eu/deliverables-IOTA/D1_3.pdf, Juli 2012, besucht Juni 2017.

[10] M. Kranz: IoT Meets Standards, Driving Interoperability and Adoption, CISCO Blog, siehe http://blogs.cisco.com/digital/iot-meets-standards-driving-interoperability-and-adoption, Juli 2015, besucht Apr. 2017.

[11] Eclipse IoT Framework, siehe http://iot.eclipse.org/, besucht Apr. 2017.

[12] Bosch IoT XdK, siehe https://xdk.bosch-connectivity.com/, besucht Apr. 2017.

[13] Relayr Plattform, siehe https://relayr.io/en/iot-middleware-platform/, besucht Apr. 2017.

[14] RIOT IoT Betriebssystem, siehe https://riot-os.org, besucht Apr. 2017.

[15] Eclipse OneM2M Implementierung, siehe https://eclipse.org/om2m/, besucht Apr. 2017.