Quality Assessment Framework for Business Processes as a Service in a heterogeneous Cloud Environment

Rupinder Vinayak
2996507
Master Computer Science
University Stuttgart

ABSTRACT

A business process is an activity or set of multiple activities that will fulfill a particular objective of the organization. Business process management (BPM) is a methodological way for the improvement of those processes. Due to increased competition in the market, companies are shifting their business processes online using some sophisticated Business process management (BPM) tools and methods.

The focus of this thesis is to design and implementation of an initial testing system for business processes which are published on a heterogeneous Cloud environment.

This thesis documents researched the state of the art of business process testing (BPT) which covers some testing techniques and selected the best-suited method for testing business process which is responsible for the quality of the system.

Also, it concentrates on the state of the art of testing the Cloud. It focus on different methodologies to test SaaS, PaaS, and IaaS. The main objective for that, is to understand the way how testing Cloud environment works, hence, it will lead to the understanding of testing of business processes as a service.

Additionally, it explains the general architecture of the TTCN-3. A design of a test system to test the business process based on TTCN-3 is presented. A case study of CloudSocket has been studied and according to the requirement, we have introduced an initial work for testing BPaaS in a heterogeneous Cloud environment. This initial test system was implemented and validated in the CloudSocket Marketplace.

TABLE OF CONTENT

ABBREVATIONS

bpt	business process testing
bpaas	business process and a service
ttc	testing the cloud
cd	coding/decoding
ets	executable test suite
mtc	main test component
pa	platform adapter
ptc	parallel test component
sa	system adapter
sut	system under test
tci	ttcn-3 control interface
te	test executable
tri	ttcn-3 runtime interface
ttcn-3	testing and test control notation, version 3
qa	quality assurance
qc	quality control
qaf	quality assessment framework
ptc	parallel test component

1 INTRODUCTION

This thesis document begins with the introduction of the topic undertaken. Background section provide the brief context of testing Business Processes as a Service. Followed by the objective, which comprise the ultimate goal of the thesis.

1.1 BACKGROUND

Initially Cloud services were just comprised of “Infrastructure as a Service” – followed by “Platform as a Service” and “Software as a Service”. “Business processes as a Service” (BPaaS) (Thomas Barton, 2014) are new addition the fleet of this series which provide business process services in a Cloud environment.

To provide such services, „quality” and a corresponding quality assessment framework (QAF) targeting the full life cycle of the BPaaS plays a very crucial role. Such a QAF serves as important element of a respective ITIL service catalogue management (ITIL SCM) (itSMF UK, 2012). The quality of the provided service needs to be assessed so that the service can meet the user/organization’s expectation.

1.2 OBJECTIVE

The main objective of this thesis is to propose, design and implement a proof-of-concept of an initial version of a testing system to test business processes running in heterogeneous Cloud environments. In this case, we assume testing as the key methodology of “model-based testing” and we will use the ETSI standard testing language TTCN-3 (TTCN-3, 2014) for this purpose.

It also introduce CloudSocket EU project which aims to provide services like business processes over Cloud environment. A corresponding test approach is designed which targets the full lifecycle of the business process provided as a Service (BPaaS) over the Cloud. This system is deployed between CloudSocket Marketplace and the customer (CloudSocket, 2016) to assure the high quality of the product provided. In marketplace, end-users can subscribe to various BPaaS bundles provided over Cloud. This testing approach tests all the processes in marketplace provided as BPaaS bundle.

The aim to design such a testing system, is to assure the quality of the BPaaS bundle to the customers. Besides, it includes the designing of an automated test framework which will also provide proper test reports and graphical logging for the executed testcases. To attain this functionality, I used TTWorkbench tool (TTWorkbench - Spirent, 2016) for writing test suites and testcases in TTCN-3.

2 STATE OF THE ART OF BUSINESS PROCESS TESTING

We begin this section by describing the state of the art of business process testing followed by detailed explanation of three different types of testing methodologies to test the business processes. The three methodologies which are discussed to test the business processes in this section are, process cycle test, decision table testing, and lights-out testing.

2.1 BUSINESS PROCESSES

A business process is a set of steps performed by a particular group of stakeholders to aiming to accomplish a concrete objective. These series of steps are performed many times repeatedly, infrequently by multiple users, mainly in specific and optimized way. A business process can be automated or manual. If it is automated than the process is achieved with technology aid which assists the clients in actualizing the process in a more precise, standardized and optimized way. And, if it is manual than the process is accomplished without the guide of an assisting technology or automation.

2.2 BUSINESS PROCESS TESTING

The procedure of verification and validation of end-to-end point of the business process is called as business process testing (BPT) (Testing Concepts, Testing Methodologies, 2016). This process is performed in a well-ordered manner to affirm that all business guidelines are working effectively and meeting business expectations. Also, any deviation and defects than the desired results are logged in Reports.

Based on the analysis of business process testing (BPT), it can be said that BPT is one of the most advanced testing technique to test business process (Testing Concepts, 2016). It lies somewhere in-between simple manual testing and complex automated testing. It promotes the automation of high-level business processes. Role-based model of BPT allows non-technical SMEs to collaborate effectively with automation engineers to increase the efficiency of the testing process.

We will focus on these below methods for the proposal of testing business processes.

2.2.1 Process Cycle Test (PCT)

The process cycle test is a method that is connected specifically to the testing of the quality characteristics for suitability. The basis of test ought to contain organized data and information on the required framework behavior the form of decision points and form of paths. The process cycle test deviates on various points from most other test design techniques and strategies:

The process cycle test is not a plan test, but rather a structure test: the test cases issue from the structure of the flow of the process and not from the outline particulars of the design specification of the test case (Kees Blokland, 2013).

The anticipated outcome of the process cycle test is very simple, the physical test case has to be executable. This verifies certainty that the individual activities can be completed. As opposed to other test design methods and techniques, no explicit expectations are made of the outcomes, thus this does not need to be checked.

PCT is a test strategy where you have to test various processes of components, where the elements are displayed by flowcharts. It is a

black box testing method and can be utilized for any framework (TMap Sogeti, 2014).

Point of focus in the five general steps

1. To identify the test situations.
2. To create the logical test cases.
3. To create the physical test cases.
4. To establish the starting point.
5. To create the test scripts.

Limitations:

Sometimes, the business flow is too complicated. In that case, flowchart method becomes complex and clumsy and due to that some bugs and errors always remain undetected. Moreover if any alterations or changes are required in later stage, whole flowchart may require to re-draw completely.

2.2.2 Decision Table Testing

It is an easy approach which helps to identify the test scenarios for complex business logic. Business rules and validations take up the major part of the requirements that are given by the customers. The requirements represented and communicated by business analysts or customers to the entire project team are presented in a logical process flow diagram. For any of the complex requirement, a rational process flow diagram has many nodes, branches and decision boxes. Testers are expected to cover all those branches touching every bits of such a complex logical tree.

It was found that the technique of testing decision table is highly useful in this aspect (Cai Ferriday, 2007) (Test strategy, Testing Methodologies, 2016). Following is an example showing how this technique helps in making the test scenario preparation for complex business logic easier. To write the test cases for a login screen with the help of decision table technique, consider the diagram below for business requirement for login:

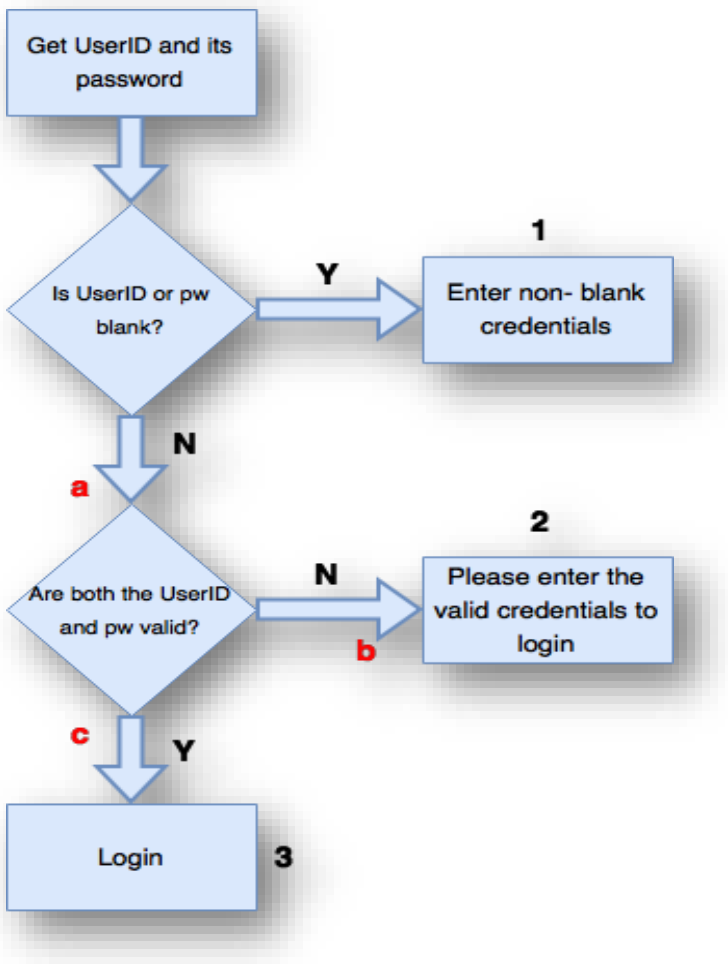


Figure 2-1: Use case for login screen (image source: softwaretestinghelp.com)

Now, on the basis of leaves (1, 2, 3) and branches (a, b, c) we will build the decision table.

Table 2.1 – Business flow decision table

Sno	Branch	Validation		Expected results	Combination
		UserID/ pw blank	UserID and pw are valid		
1	A	YES	NA	Please enter non blank credentials	Both blank either of them blank
2	B	NO	NO	Enter valid credentials to login	Both invalid either of them invalid
3	C	NO	YES	Login	NA

Advantages:

- Any sophisticated business flow represented in the form of a diagram can be easily described in the scope of this technique.
- It provides fast confidence and trust on the test cases and testers do not need to review their own test cases multiple times to gain confidence.
- Easily understandable and with the help of decision table template anyone can make test cases. It gives complete coverage at the very first shot. Hence, rework on the test case scenario can be totally avoided.

Limitations:

Some of the test case preparation techniques like boundary value analysis and equivalence partitioning cannot be directly accommodated in this template. But, can be noted down in the combination column and used while writing the test cases.

2.2.3 Lights-out testing

Test automation, particularly for bundled applications such as SAP and Salesforce are rapidly turning into a standard best practice to guarantee business processes are not disturbed by continuous enterprise technology changes. It provides a consistent production environment which helps user to get rid of undesired and costly downtime associated with defects and unavailable systems. Glitches, imperfections, and technology failures have serious impacts on customer satisfaction and the user-experience. It is still difficult to hold back on quality conformation, since testing has generally been an incredibly expensive and tedious process. Manual testing and legacy script-based arrangements require contribution from developers, analysts and experienced QA experts. “Lights out” testing method has been introduced to address these problems (Worksoft, 2015).

By automating the execution of critical routine tests, lights out testing almost eliminates the requirement for human intervention.

It yields the following key benefits:

- Ensuring error-free execution of the business process.
- Accelerating all projects by reducing the test cycle times.
- Boosting the test coverage with unaccustomed resources.
- Promoting innovation and effectiveness.
- Excluding the manual audit and compliance with the help of automated documentation.

Lights out testing is the recurring, automated, and unattended testing of pre-eminent business processes.

Each of these terms has a distinct meaning:

- Recurring - Testing is being applied to each change paying little respect to frequent occurrences. Recurring testing can be reiterated as regularly as required (daily, weekly, monthly) without tedious setup or time consumption. This has extraordinary ramifications for test outline, managing data, and execution.
- Automated - Test execution is high-speed, precise, and steady with a completely documented audit trail. Just script free automation mechanization can give satisfactory coverage inside profoundly compacted delivery cycles without extra complexity or cost.
- Unattended - Tests can be executed at any time of the day without human intervention. Unattended testing must be sufficiently flexible to handle sudden situations without prematurely ending the whole test cycle or producing spurious failures.
- Essential – Testing is centered on business forms that empower, run, or control basic organizational operations. This testing is about mitigating most extreme hazard in the briefest time possible. This guarantees the business can keep on functioning with the most elevated amounts of value and effectiveness, even when fundamental innovations, technologies, and procedures change.

Lights out testing is a demonstrated, effective approach that mitigates the innovation risk, increment profitability, and lessens the expenses of quality assurance. Most importantly, it gives organizations the certainty that each business procedure is working as per the requirements.

Customary testing strategies just give a partial solution and can't address the difficulties of a rapidly changing technology. That is the reason effective organizations in each industry are swinging to Worksoft's wise automation technology to guarantee the trustworthiness of their venture applications and each business procedure.

3 TESTING THE CLOUD (TTC)

In this section, we discussed the state of the art of testing the Cloud services. The section begins by addressing the definition of Cloud computing followed by problems with the quality of services in Cloud environment. This section of thesis also provide an overview of service categories of Cloud computing, which also includes different testing methodologies followed by respective type of Cloud service.

3.1 CLOUD COMPUTING

One of the most innovative and inventive methods which are well focused on a well centric way to the utilization of the internet & remote servers, specifically for the software application, information access, information administration. Precisely, what Cloud computing does is, it authorizes the users and organizations in such a way, to enable an easy utilization of the applications without the need of installing software or even access to any local files on their computer. Hence consequently made them utilizable on any computer, which would have the internet access of course (The NIST Definition of Cloud Computing, 2011).

3.2 PROBLEMS WITH QUALITY OF SERVICE ON CLOUDS

Undeniably, Cloud computing as well as the virtualization always lead to some real and unique challenges (Gilad, 2011) for the services and applications, which undeniably have to be well balanced for the dynamic provisioning. There would be a well-noted change in the way the applications get started and initiate the interface with the server, whenever there would be an allocation or a shift of the resources from the virtual environment or the Cloud system. The availability of hardware and software resources will be one important factor, on which it would be dependent. To give an instance, there are certain applications (Danilo Ardagna, 2014), which have hard-coded server name, which actually doesn't work in the Cloud.

Mentioned below are top three IT concerned scenarios (McNickle, 2012) which are meant for the applications in the virtual world.

1. First and foremost is the noticeable degradation of the services during the transition phase of the infrastructure precisely from Physical to Virtual infrastructure.
2. The possible negative influence on QoS of inter-application shared resource contention;
3. Dilapidation of QoS at application load times, which are peak usually

There are some more impactful factors (McNickle, 2012) which eventually affect the quality of the services of the Cloud & its performance which includes:

(a) Increment in the requirement of the Cloud-based system
(b) Enabling the ensured customer satisfaction & quality
(c) Remote testing

(d) Infrastructure

(e) The situation, when the providers for Cloud computing become capable of more control

3.3 SOFTWARE AS A SERVICE (SAAS)

3.3.1 What is SaaS

SaaS, Software as a Service is the software, which is being deployed and provided over the Cloud environment (Grance, 2009). It represents one of the biggest Cloud markets and is growing exponentially fast (Rouse, 2016). The web is being used so that applications can be delivered, which are thoroughly managed and organized by a third party vendor and whose interface is undeniably accessed on the client's side. The majority of the SaaS applications can be run directly from a web browser, needless of any downloads or installations. Some plugin's requirement would be there, though. Blaming to the web delivery model, there is a well-noted eradication of the need for installation and process of application's running on individual computers by SaaS. Some imminent examples of SaaS are: Google Apps, Salesforce, Workday, Concur, Citrix GoToMeeting, and Cisco WebEx

3.3.2 What kind of testing methods are available for SaaS?

Various kinds of testing elements which are lying within the scope of old traditional testing techniques can be well eliminated from the techniques of SaaS Testing (Mengerink, 2013) (Apprenda Inc, 2016). There lies no requirement for testing of the cases for: Server or Client installations, multi-platform back-end support, multiple version support or backward compatibility.

In order to speed up the application testing, the SaaS testing methodology incorporates agile methods specifically, so that to make SaaS Vendor's services more rapidly delivered to the market.

As a matter of fact, SaaS has the capability of employing automation tools, to handle any specific QA (quality assurance) tasks, which precisely includes: Unit testing, functional testing, testing of SOA interfaces and performance testing of the application which is running on the platform of Cloud services.

3.4 PLATFORM AS A SERVICE (PAAS)

3.4.1 What is PaaS

As clear from the abbreviation, it stands for Platform as a service (PaaS), which are majorly used for applications and another kind of development, while the provision of the Cloud components to software is done (Grance, 2009). So what exactly the developers achieve as an advantage with PaaS, is precisely a well-defined framework which they can build upon to develop or customize applications?

PaaS possess the capability of making the development, testing and deployment of applications faster, subtle and more profit oriented .i.e. cost effective! (Cloud Standards Customer Council, 2015) With the advent of this technology, enterprise operations, or a third-party provider gains the capability of managing OSs, virtualization, servers, storage, networking, and the PaaS software itself. Developers, however, manage the applications (Apprenda Inc, 2016).

3.4.2 Which Testing Methods are available for PaaS?

There are various testing methods by which we use to test the PaaS environments (Dr. Rahul Malhotra, 2013):

- Elasticity Testing.
- Security Testing: Security testing which is a crucial aspect of testing applications due to increase in the security breaches in business. This has the potential of providing an assurance that business critical data is stored & transported securely.
- High Availability (HA) and Performance testing: Used to test the behavior of an application or a system when subjected to amplified load from multiple locations.
- Compatibility testing: It inculcates testing user interfaces of the Cloud based application, and validating its compatibility when migrated to a Cloud platform.
- Multi-Tenancy Testing: The main emphasis of this testing is to confirm that users from different organizations become capable of using the Cloud services. It also involves validating the Cloud offering such that the service is customized for every client's requirement.
- TMap and TPI (Sogeti GmbH, Capgemini, 2014).

3.5 Infrastructure as a Service (IaaS)

3.5.1 What is IaaS

Infrastructure as a Service (IaaS), is a kind of self-service model where one can access, monitor, and manage remote datacenter infrastructures, such as compute, storage, networking, and networking services (e.g. firewalls) (Grance, 2009) (The NIST Definition of Cloud Computing, 2011). Instead of buying hardware outright, users can purchase IaaS based on their consumption and need, similar to electricity or other utility billing.

As compared to SaaS and PaaS, IaaS users are responsible for managing applications, runtime, data, OSs', and middleware, (Apprenda Inc, 2016). Providers of IAAS, still manage virtualization, hard drives, servers, storage, and networking.

IaaS Examples: Amazon Web Services, Microsoft Azure, Cisco Metapod, Joyent, Google Compute Engine (GCE).

3.5.2 Which Testing Methods are available for IaaS?

Testing is important, not only for software developers but also for the people working with IT operations. These below are few point which could be considered while testing an IaaS (Bertram, 2015):

- **Workloads are being tested**: All the Cloud services or an on-premises processes?
- **Performance considerations**: Do we need 100 Input/output Operations per Second (IOPS) or more like 100,000 IOPS?
- **Commitment to Time**: How much time do we want to invest?
- **Availability**: Do we need to access it from everywhere?

4 TTCN-3 IN GENERAL

The testing and test Control Notation version 3 (TTCN-3) is a test specification and test implementation language created by industry and academia experts at the European Telecommunications standards Institute (ESTI) (TTCN-3, 2014). It is a powerful scripting language and has been successfully employed in testing composite applications enabled in SOA.

TTCN-3 is easy to use as a standardized testing language which has a look and feel of a regular programming language with comparatively less

complexities. The main objective behind designing TTCN-3 is to provide well-formed easy to understand test case definition independent from any application domain. It mainly comprises well-defined operational semantics, robust built-in matching mechanisms, it support timers, and is able to configure runtime parameters for the test execution. It also supports message- and procedure-based communication to exchange messages with the system under test (SUT). TTCN-3 can be used in many type of testing namely, validation testing, unit, integration, software module testing, functional and load testing, regression and approval testing, etc.

The development of a TTCN-3 test system (TTCN-3, 2014) requires:

- TTCN-3 test suite
- TTCN-3 Compiler
- Implementations of testcases in TTCN-3 using TTWorkbench
- Implementation of a Codec for encoding and decoding of the data send/receive from the SUT.
- Implementation of SUT Adapter for establishing the required communication channel with SUT interfaces.

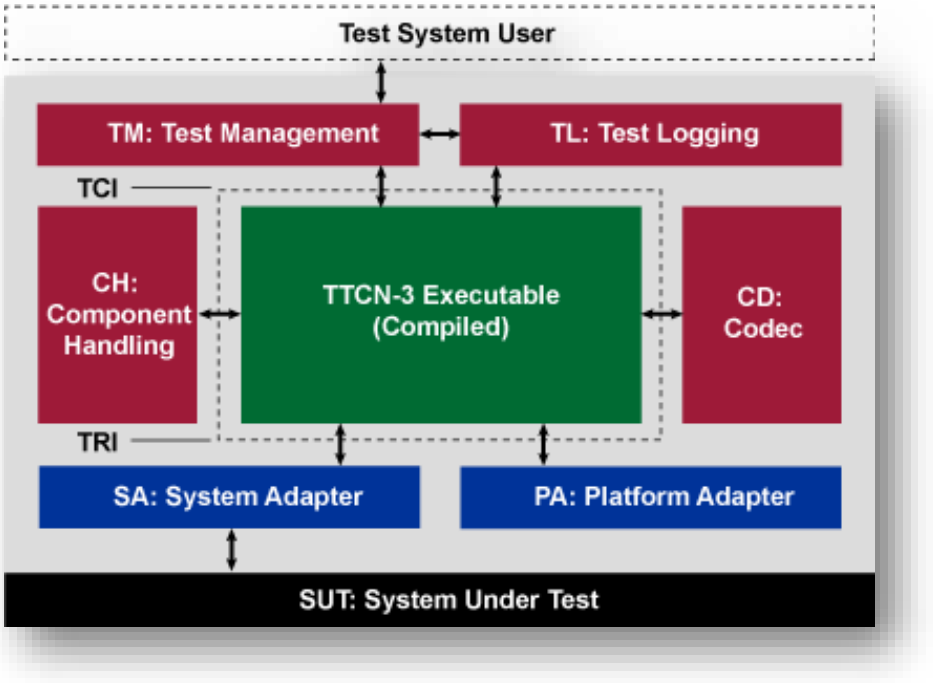


Figure 4-1: General TTCN-3 Architecture (Testing_Tech, 2016)

The standardized interfaces like TCI and TRI make it feasible for a TTCN-3 test suite to be adjusted, so that it can be execute in a scope of various scenarios, platforms, operating systems, and environments. Besides, it encourages testing in early developmental period of SUT because, for instance, the communication channel can be adjusted to the currently available system.

TTCN-3 is used to indicate tests in a dynamic and abstract way, however, a test framework is required to execute these tests. A TTCN-3 test framework involves elements that acquire conceptual details of TTCN-3 into concrete implementation of the test framework. For instance, virtually characterized communication is mapped to real physical/real communication channels. In this thesis, the focus is on two major interface between TTCN-3 test framework and system under test. These are ttcn-3 runtime interface component system adapter and ttcn-3 control interface component Codec (Testing Technologies IST GmbH, 23 November 2015).

The TTCN-3 Executables are the part of the test system which depicts how it manage interpretation and execution of the test system. This can be correlated with the test framework implementation environment to produce and compile the code by the TTCN-3 compiler TTWorkbench. The rest of the part of TTCN-3 test framework, which manages any perspective that cannot be concluded from the information being available in original module, can be decomposed into test management, System Adapter (SA), etc. (see Section 8.2)

A TTCN-3 test framework has two noteworthy interfaces, TTCN-3 Runtime Interface (TRI) component System Adapter and TTCN-3 Control Interface (TCI) component Codec (Figure 4-1) (TTCN-3, 2014).

TTCN-3 Control Interface (TCI)

Test management (TM): It is responsible for the general administration and management of the test framework, user interface for the test system, execution of tests and management of runtime parameters.

Test Logging (TL): This part belongs to the management of logging the events and exceptions of the test system.

Coding and Decoding (CD): The role of Codec is to understand the type and structure of the payload which is received from the SUT via system adapter. It fetches the data and starts decoding it with the help of using respective parsers used according to the type of expected payload. Here templates for the expected payload can also be defined to match the quality of the System under test.

Component Handling (CH): It plays the role of the channel which is used to handle the communication between two or more test components.

TTCN-3 Runtime Interface (TRI)

System Adapter (SA): The role of system adapter is similar to the network adapters and it is used to connect the test component with the system under test to get the information from SUT in the form of payload. System adapter basically takes the information from the SUTs with the help of defined transmission control protocols under some set of rules.

Platform Adaptor (PA): It executes the TTCN-3 external functions and gives framework the single time notion.

TTCN-3 has a fundamental model of separation of concerns which is clearly depicted in the Figure 4-1. This separates the Abstract Test Suite (ATS) from the coding/decoding and communication, and presentation details; providing powerful matching mechanism that separates behavior and the conditions governing the behaviors and there by promoting a systematic approach to testing. This separation of concern provides full portability of test suites, making them independent of platform implementation.

TTCN-3 provides powerful re-usable test oracle specification language construct called a template and corresponding built-in matching mechanism for test oracles verification. The template concept is based on data structure and allows for the specification of multiple test oracles in one single operation. The TTCN-3 concept of a template is a double-edged concept that is used both to define data to be sent to an SUT and test oracles to be matched against incoming messages. A template resembles a structured variable data assignment as follows:

```
template urlType urlTemplate := {
    protocol := "http://",
    host := "www.mocky.io/",
    file := "v2/582dcebd110000bd0fd76122"
}
```

The above template can be used to send its data to the SUT using the TTCN-3 send keyword as follow:

```
    httpPort.send(urlTemplate);
```

It can also be matched against incoming messages using the TTCN-3 receive keyword as follows:

```
httpPort.receive(jsonTemplate)
```

This concept allows powerful re-usability compared to the usual assertions of JUnit for example. The TTCN-3 concept of template actually enables the separation of concerns between test behaviors and condition governing behavior. This avoids the usual pitfalls of spot checking with its trail of errors and omissions. TTCN-3 also provides a wide selection of tools that offer powerful test execution results inspection features that enable the tester to focus on individual elements of a web page.

TTCN-3 is an international standard that was originally designed for testing telecommunication protocols which consist of discrete messages between communication entities. It is a strongly typed test scripting language which was successfully employed in conformance testing of

communicating systems, where in discrete events like sending/receiving of data units of protocols and abstract service primitives are used. Conformance test suites are developed based on the protocol specification. TTCN-3 is also employed in testing avionics systems that send or receive huge series of periodic messages with an identical payload at precise time intervals (TTCN-3, 2013).

TTCN-3's matching mechanism enables multi-user matching of request-responses with precise detection and location of faults and quality of service issue. The test specification approach used in TTCN-3 supports parallel testing and promotes reusability of data types tied to reusable components.

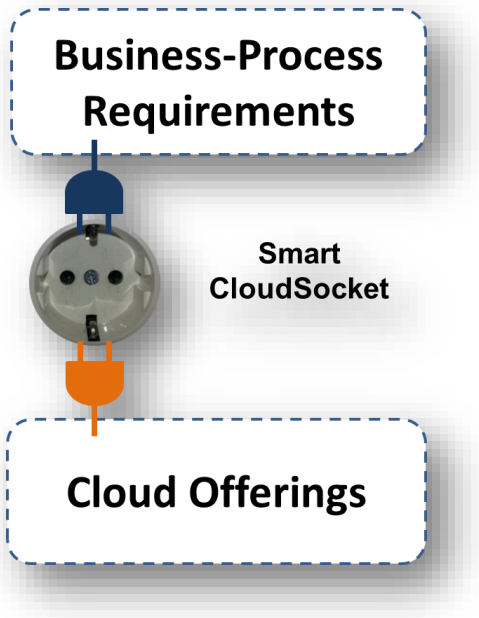
5 APPLYING TTCN-3 TO BPAAS TESTING BASED ON CLOUDSOCKET

This section will be concentrating principally on introduction of the concept of CloudSocket. Here, this document describes the definition of CloudSocket project and its vision. Then, the architecture of CloudSocket is explained which includes four essential blocks of BPaaS lifecycle in CloudSocket project. Lastly, this section also provides an overview of the part of the BPaaS lifecycle architecture which is related to this thesis.

5.1 CLOUDSOCKET AS A PROJECT

CloudSocket is the name given to the project which aims to provide business processes to organizations through its online marketplace over Cloud (CloudSocket, European Union's Horizon 2020 Framework Programme). The objective of the project is to encompass a "hybrid

process" modeling system that reconciles semantic derivation, protocol based induction, meta-modelling management methods and information management methodologies. Hence, conveying SMEs closer to the Cloud by creating them attractive for them to consolidate Cloud resources and component for the acknowledgment of the objectives. The proposed system executes a layered approach for managing the complexity of bridging the semantic separation from business process to the configuration business process work flows in the Cloud.



"Business Process as a Service" (BPaaS) is considered as another new concept that is being presented as the successor of the Cloud concepts after IaaS, PaaS, and SaaS. Consequently, BPaaS is not seen as a technical blend of SaaS but rather is viewed as an intermediate type between IT-agnostic business clients and Cloud Computing (CloudSocket, 2015). Business process management is a very well established approach based on modelling (Frank Leymann, 2000); henceforth CloudSocket utilizes business process models as the sources of IT-Cloud requirements. This project of CloudSocket, advances the possibility of a "Hybrid process" modelling framework with the help of already known techniques for rule-based inference, meta-modelling, semantics and knowledge management techniques which fills the gap between business requirements and exploitation of Cloud resources and components.

The proposed structure actualizes a layered approach for dealing with the multifaceted and complex scenarios to bridge the semantic separation from the business process to workflow design the configuration of BPaaS in the Cloud (CloudSocket, 2015).

5.2 CLOUDSOCKET: ARCHITECTURE

The growing demand of the business processes over Cloud has increased the demand for BPaaS testing as well. However, there are multiple challenges that are experienced in the whole course of Business Process as a Service Testing (BPaaST). For example, designing a test system for testing the business processes over the Cloud environment. But rather than discussing the challenges and their solutions, firstly we should be aware of the big picture of the implementation and architecture of the CloudSocket project. Basically to have a deep understanding of the concept of CloudSocket.

The CloudSocket technical infrastructure indicates the four BPaaS environments relating to previous ideas and concepts. Whereas the focus is on the interaction between the distinctive scenarios and the expected results, as those environments and scenarios can be acknowledged by different implementations. Technical project partners realize and understand each BPaaS environment, however, the CloudSocket can likely be built up with other set of tools, as long as the interfaces between the BPaaS environments are obeyed.

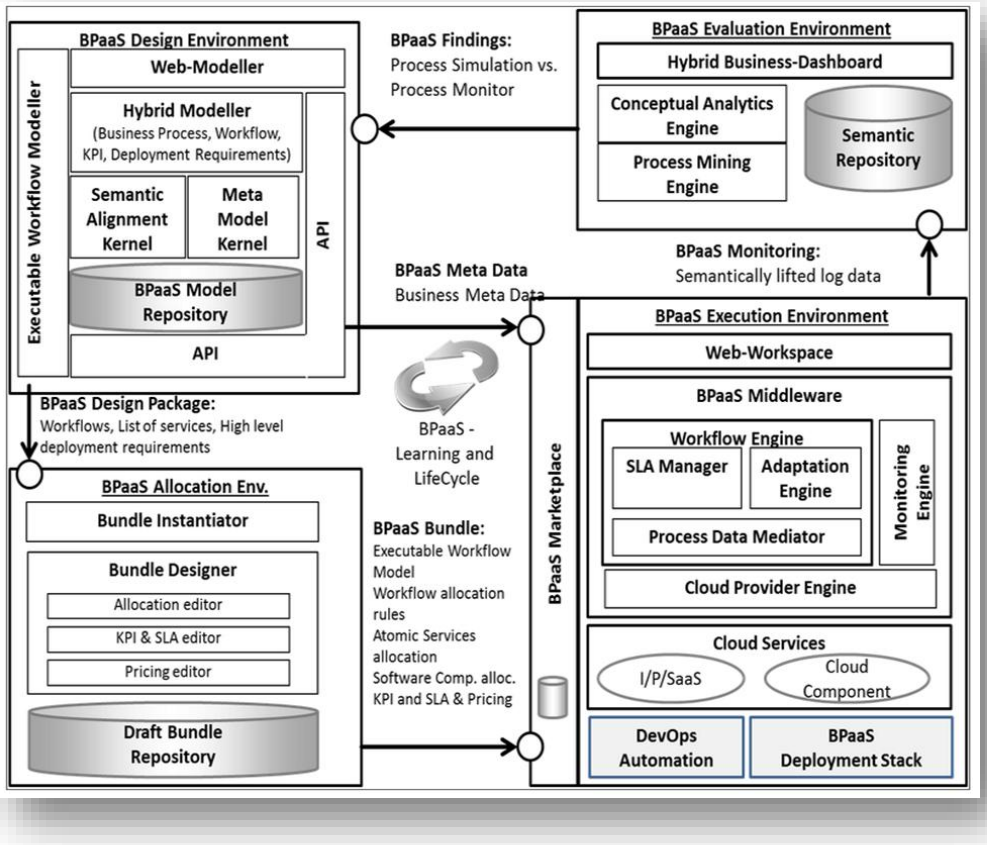


Figure 5-1: CloudSocket Architecture (CloudSocket Consortium, 2015)

The technical infrastructure of CloudSocket includes four primary building blocks/environments which guide to the four unique phases of the business process lifecycle.

The figure (see figure 5-1) visualizes CloudSocket's initial design where every environment, not just only concentrates alternate business process lifecycle, additionally endeavors to provide the appropriate support empowering business and IT alignment (CloudSocket Consortium, 2015). It should also be highlighted that apart from the four principle environments, there is also the *BPaaS MarketPlace* which goes about as an outer interface of the BPaaS *execution environment* giving fundamental marketplace functionality to CloudSocket agent clients that incorporate the visualization, purchase, and deployment of BPaaS offerings. Finishing, we quickly examine the fundamental basic

usefulness and functionality of each environment by also indicating which component delivers which part of the primary functionality.

In the first block (see Figure 5-1) we can see there is a *business process design environment* containing different kinds of business processes to be served over the Cloud as a service with the help of incubators. It gives the modeling and mapping functionality to CloudSocket agents empowering them to plan applied business process and also map them to the specifications of the executable workflows that acknowledge them.

Next, The *BPaaS Allocation Environment* goes for bundling the executable workflows into a Cloud-deployable workflow package which can then be given as another BPaaS offering by the *CloudSocket Marketplace*. Here, things like Service Selection, Service discovery, etc. will take place.

At the point when a specific BPaaS offering is bought by means of the Marketplace, the *BPaaS Execution Environment* is responsible for conveying it in the Cloud via Cloud Provider Engine, thus bringing it into operations. Here Marketplace is modeled and designed, and with the help of BPaaS modeling, the marketplace is realized on the Cloud with different rules.

The *BPaaS Evaluation Environment* is the best place for BPaaS performance analysis and improvement. It offers representation mechanisms via Hybrid Business Dashboard which not only permits demonstrations of KPI assessments but also their drill-down into lower-level KPI and metric assessments.

5.3 PART OF ARCHITECTURE RELATED TO THIS THESIS

A client of CloudSocket browses the Marketplace and chooses a specific BPaaS application package through the support of the marketplace assistance framework. A CloudSocket customer buys this package and makes some Cloud services incorporated into it. Once these activities are completed, the package is deployable and hence sent to the *Execution Environment* for adaptive provisioning and operation.

The CloudSocket *Execution Environment* empowers managing, monitoring, and adapting the execution of the BPaaS packages produced in the allocation block, which has been published over a heterogeneous Cloud via Marketplace. So, all the BPaaS offerings presented in Figure 5-1 are provided to clients via Marketplace along with Execution Environment which empowers their operations in the Cloud (Figure 5-2).

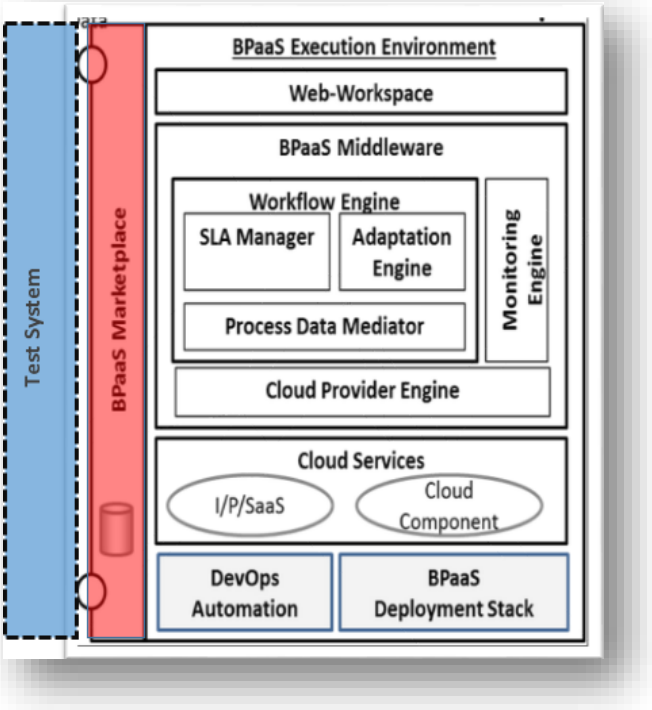


Figure 5-2: Highlighting the modules for Test System and Marketplace in BPaaS Execution Environment.

The role of the marketplace is to publish the BPaaS bundles so that the customers, organizations, Cloud providers, and the systems can purchase. To increase the efficiency of the published content, BPaaS bundles should be thoroughly tested before going live on the Cloud via *Marketplace*. The *marketplace* is the System under Test in this thesis. In figure 5-2, you can see the additional Test System added just before the marketplace to achieve the goal precisely.

6 BPaaS Testing Approach for CloudSocket

This section describes the need of BPaaS testing approach in the CloudSocket environment. Followed by the architecture of the designed test system implementation approach for prototyping and designing of the Test System in CloudSocket environment. Next, this document describes the architecture of designed test system for BPaaS.

6.1 BPaaS Testing Approach

CloudSocket Marketplace manage: publish, subscribe and purchase process of business process bundle for the customers (CloudSocket Consortium, 2015)(Section-5.1). This process depends upon few other phases of business process lifecycle (see figure 5-1), the earlier system of the CloudSocket architecture was lacking with the dedicated framework for quality assessment of business processes. Before the thesis started, requirements for the testing data were captured in natural language documents. The absence of a testing framework reduces the confidence of the published business processes if they meet the desired expectation of the customer or not. All the teams working on their respective modules were testing the systems at their end only.

Testing BpaaS (BpaaST) ensures high quality across the deployment lifecycle. It also includes the security, privacy, accessibility, and standard compliance as well. The automated validation and testing of the functional and non-functional aspects of the business process as a service help shorten the release cycle of the frequent upgrading application. The key to successful BPaaST (testing Business process as a service) is putting together the right combination of test strategies, automate the tests for functional and non-functional requirements, and leveraging the best practices that would help maximize the client investment. In intend to achieve company's ultimate goal of business outcome.

6.2 ARCHITECTURE OF DEVELOPED BPAAS TEST SYSTEM

At this point I commenced the structure to test the business processes before publishing over BPaaS bundles to CloudSocket Marketplace. Any BPaaS bundle should be tested to increase its efficiency of deployment. Newly developed TTCN-3 test framework is induced between end-user and *CloudSocket Marketplace* (see figure 6-1), so that every BPaaS bundle will be thorougly tested before the subscription for the service starts. The main components of this designed test system are, TTCN-3 test framework, System adapter, and Codecs.

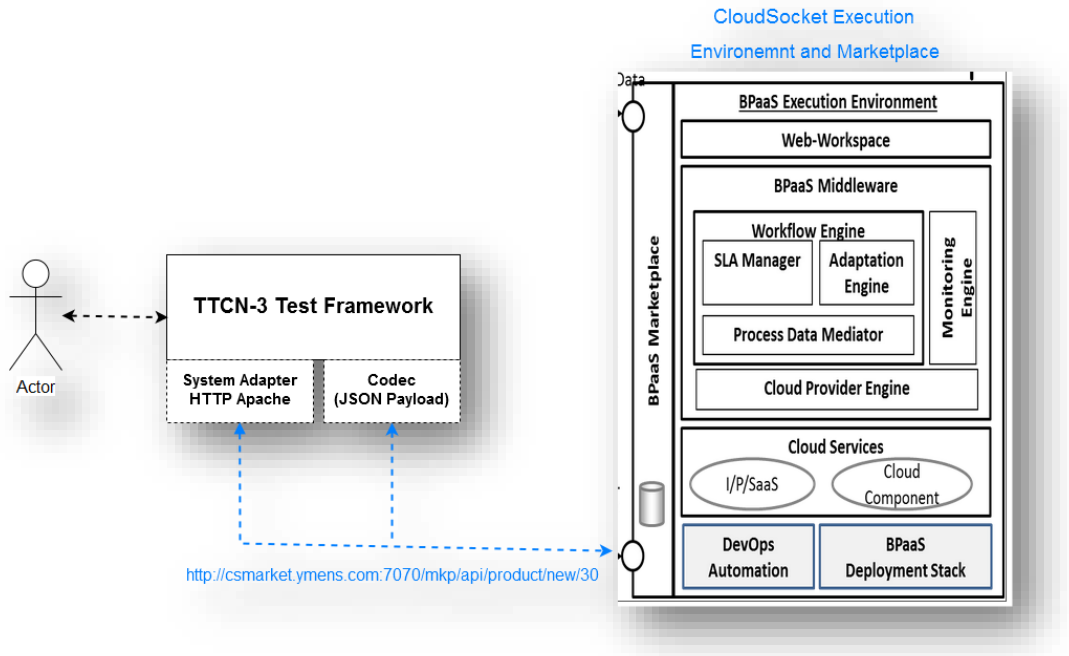


Figure 6-1: Developed TTCN-3 Test Framework

The test system gets the data from *CloudSocket Marketplace* via HTTP Apache System Adapter, and decode it for further manipulation. Received data is then compared with the data which is already defined in TTCN-3 test framework expected data templates to match the quality. If the received data matched with the expected data, the test will be passed, else failed.

The objective of the thesis is to design an initial testing approach for the business process bundles kept on CloudSocket Marketplace (as depicted in figure 6-2).

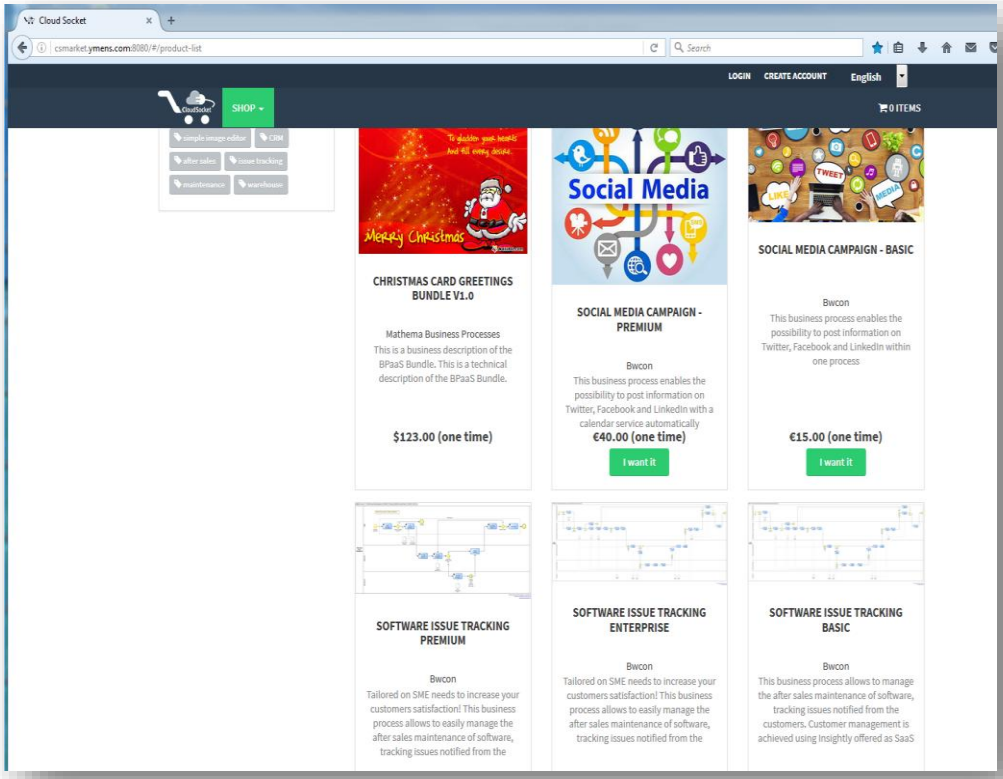


Figure 6-2: BPaaS under test: CloudSocket Marketplace; (Image source: http://csmarket.ymens.com:8080/#/product-list)

For instance, a business process from the CloudSocket marketplace (has been imported. In diagram (see figure 6-3) the case for the multichannel digital media marketing business process has been taken into consideration as the system under test to assess its quality from our QAF.

Test components receive the test data of this type of business process from CloudSocket servers in the form of JSON payload. Here, newly designed QAF plays a crucial role to match the received payload to pre-defined expected payload according to the test case.

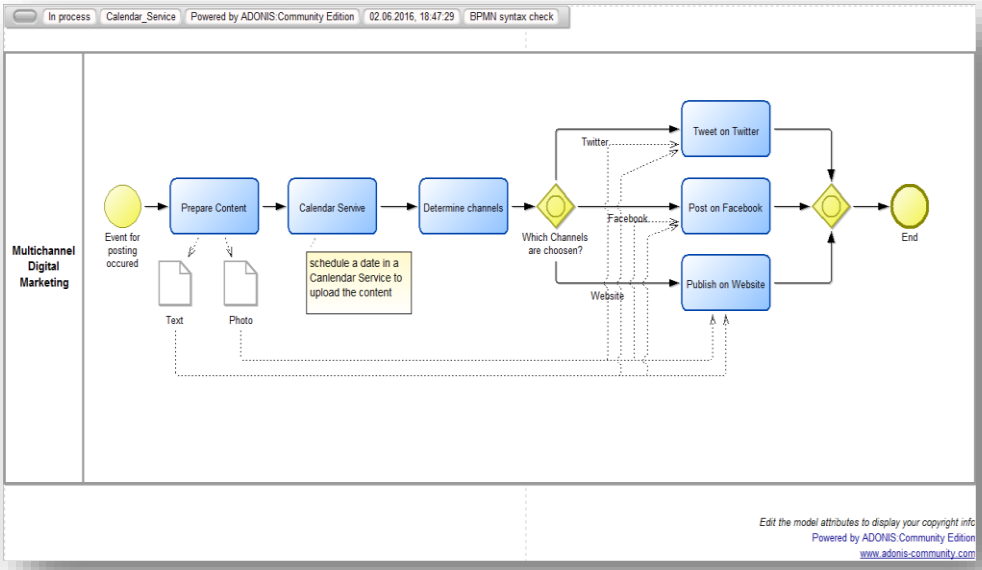


Figure 6-3: Business Process Model: Multichannel digital marketing

For the above-depicted Business process, we used TTCN-3 to verify and validate the quality of the data received from SUT csmarket servers through system adapter (HYRKKÄNEN, 2004) and codecs.

7 TTCN-3 FOR TESTING CLOUDSOCKET BPAAS

Testing and Test Notation (TTCN-3) is an internationally standardized language for testing the system by defining test scenarios. It is developed purely for testing purpose by European Telecommunication Standards Institute (ETSI) in 2000 (TTCN-3, 2014). Here, this thesis presents the prototyping and realization of the designed test system using various components of TTCN-3. This section begins with the explanation of the problem description of our case study of CloudSocket. Followed by the detailed description of writing *testcase* using *TTWorkbench Development Perspective*. And explains about the test execution process used in *TTWorkbench Execution Management Perspective.* Then, this

section provide a detailed overview of the *System adapter* and the *Codecs* which are developed exclusively for this test system.

7.1 PROTOTYPE AND REALIZATION

In the implementation of the CloudSocket project, there were several services which have to be executed simultaneously to test the quality of the designed framework and business processes. To execute these test component in-parallel, TTCN-3 is used to verify the multi-service orchestration testing verification of the results at critical points throughout the architecture. To achieve the quality of the product/service to be given over Cloud, few test measures are done with the help of designed test cases in TTCN-3. Testing is performed to ensure BPM system provides maximum desired quality to the customer with correct results. A detailed example of the TTCN-3 implementation of CloudSocket Marketplace in TTWorkbench can be seen further in the section. It also consists of some powerful features like graphical logging, which enable users to see the test results in more interactive manner. In addition to this, summarized report of the test results is also available for the entire execution of the test. TTCN-3 editor like TTWorkbench has different perspectives for programming and management of the project. First is “TTCN-3 Development Perspective” (Testing Technologies IST GmbH, 23 November 2015) view where the user can write/update the source code for TTCN-3 test cases, System adapter, Codec, etc.

Second is “TTCN-3 Execution Management Perspective”, where the user can execute the designed testcases in an interactive interface. (Testing Technologies IST GmbH, 23 November 2015)

7.1.1 TTCN-3 Development Perspective

All the development and implementation of source code related to the TTCN-3 project, either it is TTCN executable source codes or Java implementations of System adapters or Codecs is done in TTCN-3 Development Perspective.

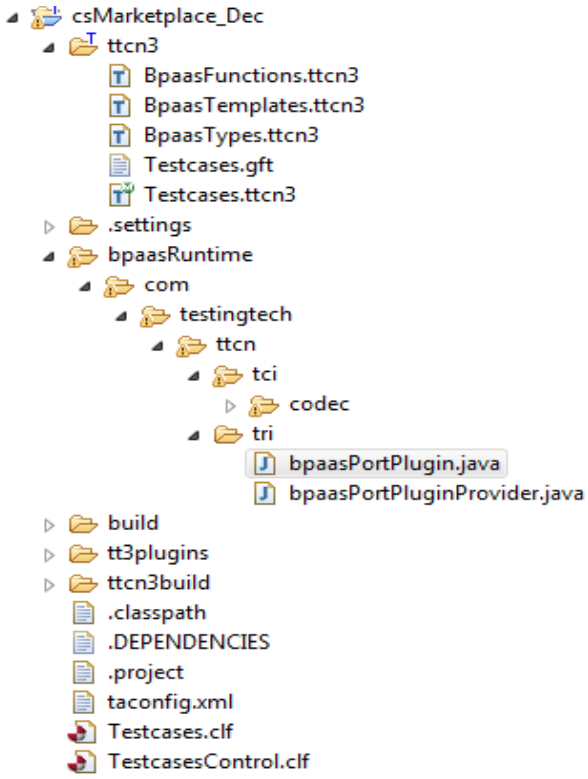


Figure 7-1: TTCN Project Explorer

A typical view of TTCN-3 project explorer in TTWorkbench looks like as depicted in figure 7-1 on the left (TTWorkbench - Spirent, 2016). For each test case component in the given testSuite, a TTCN-3 test case module is produced including templates to be sent and received and behavior definitions for the client and server sides. Any TTCN-3 module can be categorized with different module declarations. There can be data types, template, function, test case, etc. in a typical TTCN-3 executable (Testing_Tech, 2016).

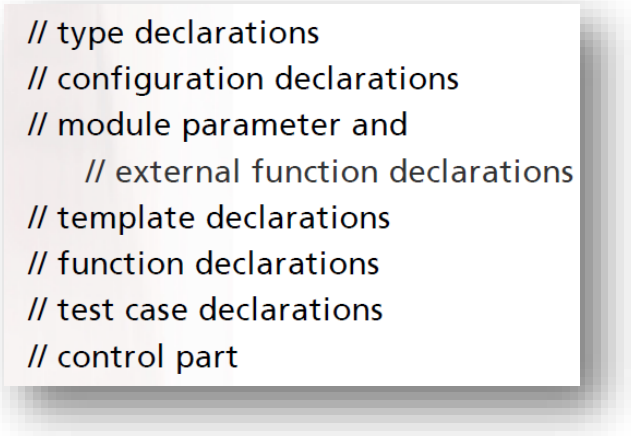


Figure 7-2: Typical content of a TTCN-3 Module

Different type of declarations can be seen in any particular TTCN-3 executable.

- //type declaration: this part of a program consists of a user-defined code, which defines data types (messages, information elements, set of various elements). "type" can be a *record, a record of, set, set of,* etc. We have defined the structure of our expected payload here using *"record of"* and *"set of"*.
- //configuration declaration consists of the definition of the communication protocols for the test components and ports. Our port type for *httpport* is defined under this part of code. By defining our http component here, we can now use the protocol to send/receive the messages and information from the system under test (SUT).
- //module parameter and external function declaration, basically define the execution configuration for the source code.
- //template declaration is the design and the structure of the data which is being sent/received from SUT. In order to test the actual data received through the port, encoding has to be done in a way that TTCN runtime environment can understand the structure

and syntax as mentioned in this template declaration. We will be accessing the csmarket servers to extract the payload for the quality assurance. Templates declarations contain test data, which is transmitted or expected during the test execution.

- //function declaration is used to determine the test behavior of the TTCN-3 test case. Furthermore, declarations, timer operations, and statements can also be integrated into the function. Functions can also call other functions recursively to do particular tests according to the requirement and the design.

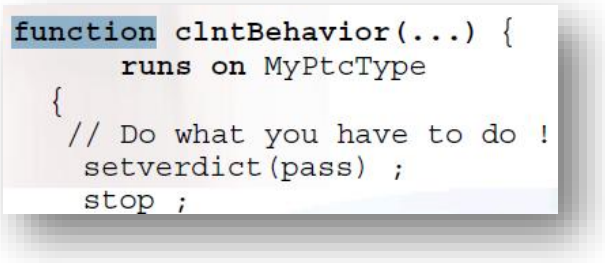
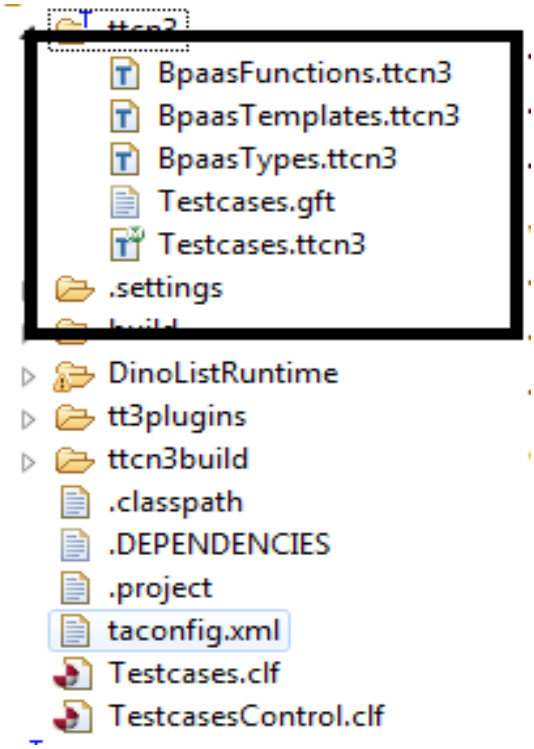


Figure 7-3: TTCN-3 Function declaration

- //testcase declaration: Syntactically, any *test case* function should start from keyword test case followed by the complete signature of the test case. The test case speaks to the dynamic test behavior and can generate more parallel test components. Similar to functions, test cases may also contain a declaration, statements, timer functions, function calling operations, etc. In our case, the test case is executing the function ptcBehaviour() according to the number of parallel test components (PTCs), NUMBER_OF_PTCS is a macro which is usually defined in starting of the code. The role for the function ptcBehaiviour() is to compare the decoded JSON input stream to the template values already given for Quality assurance.



All these declarations can be put into one single file for simpler and small projects. But if the project is more complex (for instance CloudSocket) and comparatively larger, then it is good to follow modular approach for programming. Also, calling different modules as a function of other modules as per requirement of the code. A modular approach for our TTCN-3 files can be seen in the image on left where, Functions, Templates, Types, and Test cases, all are defined and declared in separate modules.

7.1.2 TTCN-3 Execution Management Perspective

TTWorkbench is a full-fledged tool which integrates test development and Test execution management perspective in single IDE. For running and executing the designed tests, we may require particular environment runtime variables which can be configured only in a specific environment.

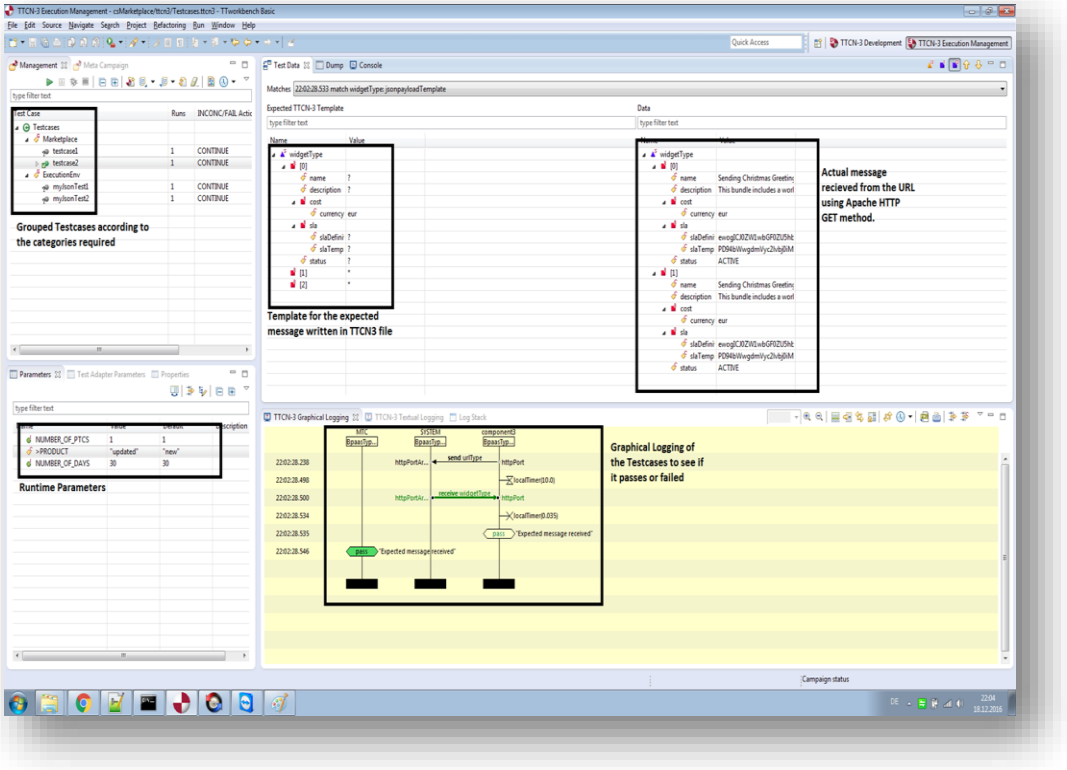


Figure 7-4: Screenshot depicting different views in the Execution Management Perspective

TTWorkbench provides us with that type of environment called as TTCN-3 Execution management perspective. Here, we define/re-define the existing variables to be tested under runtime variables. As depicted in the Figure 7-4, TTCN-3 Execution Management View is buildup with 7 different views:

- The management view is the focal perspective of TTWorkbench Execution Management perspective. It gives an interface to the user to choose a test suite to initialize and abort the test execution. All operations in the view will be then with respect to the chosen test suite.

- The parameter view permits the user to view and edit the runtime variables.
- TTCN-3 graphical logging view shows the output of test execution in a graphical way.
- TTCN-3 Textual logging shows the same test execution output view in textual format.
- The test data view is used to show the data exchange between test components and SUT during test execution.
- The dump view is used to display the data exchange as hex or plain text.

We can access the Execution management perspective by just double clicking *.clf in a project explorer from development view. Test campaign loader file (*.clf file) is an executable file which is used to execute the test suites. We can write and edit these files according to our requirement. I have created my own clf file which contains a declaration of different runtime parameters and test case groups. As seen in Figure 7-5, test cases are grouped according to the type of there execution.

This grouping in clf file can be seen in management view in execution management perspective. See Figure 7-5:

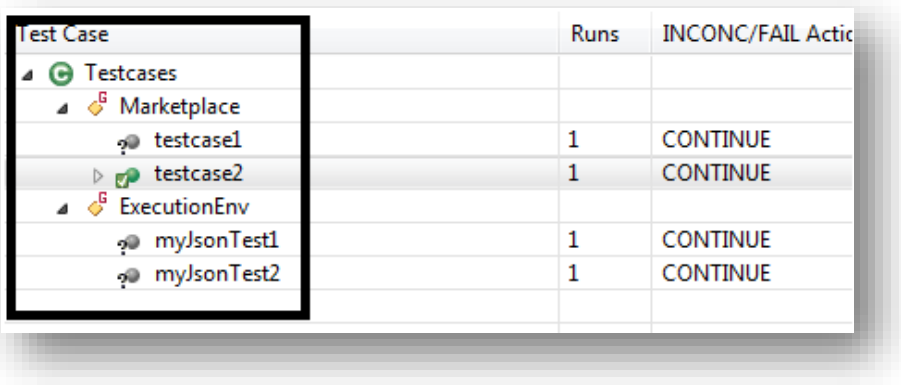


Figure 7-5: Grouped Testcases according to their categories

Runtime parameters in execution management perspective can play a vital role in test execution, as the user can do real-time changes to the parameters. For example, the user can increase or decrease the number of PTCs to be used during the test, in this thesis, I have written the code accordingly so that user can also enter the number_of_days into runtime parameters to get the data from SUT for that particular number of days. Similarly, if the user wants new or updated data, this can be also mentioned directly in runtime parameters during test execution. You can see the runtime parameter view and declaration of runtime variables in clf file in below Figure 7-6.

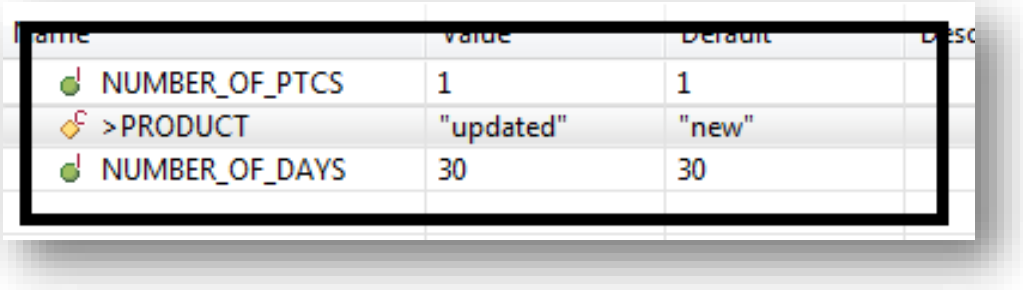


Figure 7-6: Runtime parameter view in Execution management perspective

7.1.3 Developed System Adapter

System adapter is integral part of TTCN-3 runtime environment (HYRKKÄNEN, 2004). It is a framework which basically concentrates on the interface with which SUT System adapter communicates and interact with the rest of the test component system.

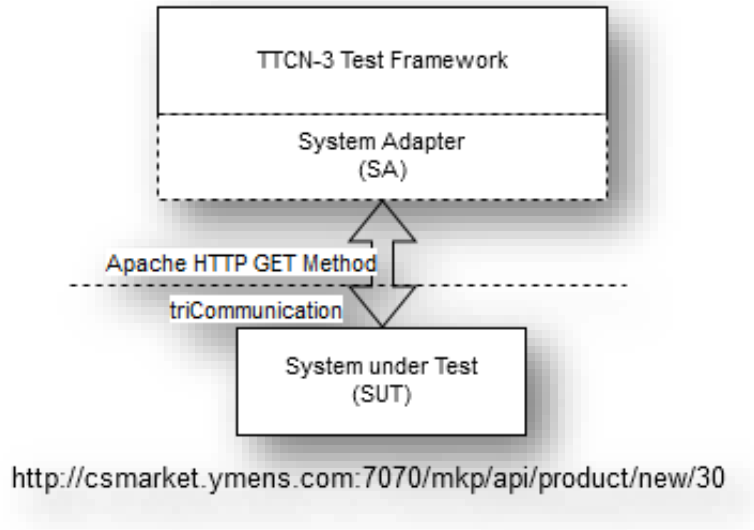


Figure 7-7: TTCN-3 Runtime Interface (TRI), showing System adapter and triCommunication

```
CharstringValue protocol = (CharstringValue)((RecordValue)sendMessage).getField("protocol");
String str_proto = protocol.toString().substring(1, protocol.toString().length()-1);
CharstringValue host = (CharstringValue)((RecordValue)sendMessage).getField("host");
String str_host =  host.toString().substring(1, host.toString().length()-1);
CharstringValue file = (CharstringValue)((RecordValue)sendMessage).getField("file");
String str_file =  file.toString().substring(1, file.toString().length()-1);
CharstringValue product = (CharstringValue)((RecordValue)sendMessage).getField("product");
String str_product =  product.toString().substring(1, product.toString().length()-1);
//CharstringValue day = (CharstringValue)((RecordValue)sendMessage).getField("day");
//String int_day =  day.toString().substring(1, day.toString().length()-1);
IntegerValue day = (IntegerValue)((RecordValue)sendMessage).getField("day");
Integer int_day = day.getInt();

String link  = new String(str_proto + str_host + str_file + str_product +"/"+int_day);
```

Figure 7-8: Merging different elements to generate *Link* variable

System Adapter is a bit of source code that handles the real communication between the system under test servers and the programs that run test cases like test components and decide their test results. It realizes the message- and procedure-based communication with SUT. In this implementation, we can see the static connection is being made between test components and SUT via Apache HTTP communication protocols.

In System Adapter, I mentioned the rule for establishing the connection by using Apache HTTP GET method. (Apache HTTP Project, 2005-2016).

To put it briefly, System adapter (SA) is responsible for the actual communication of test components with the SUT. In other words, we can mention that the real message exchange between test component and SUT is happening under the scope of SUT System adapter.

```
/*********Http based new addition in code*********/
CloseableHttpClient httpclient = HttpClients.createDefault();
//HttpGet httpGet = new HttpGet("http://echo.jsontest.com/one/two/key/value");
HttpGet httpGet = new HttpGet(link);
CloseableHttpResponse response1 = httpclient.execute(httpGet);
System.out.println(response1.getStatusLine());
//httpGet.releaseConnection();
/*************************************************/
```

Figure 7-9: Apache HTTP GET Method in System Adapter (SA)

The runtime interface between test component TTCN-3 executable and SUT System Adapter (SA) is called as a triCommunication interface (Testing Technologies IST GmbH, 23 November 2015). It is generally written and implemented in Java. As mentioned earlier, the majority of the implementation of the transmission methods used, are mentioned in SA. Any form reference to triCommunication or TTCN-3 Runtime interface basically means the interface between TE and System adapter. SUT is able to send a message to test component via System adapter using *send* statement. It is done by calling triSend() function. In this thesis, I have implemented system adapter using Apache HTTP Get method. When SA receives a message from SUT over mentioned method, it starts streaming all the messages in an output buffer to send further to Test component. It can forward the messages received to TE by just calling the operation triEnqueueMsg(). When SA call the triEnqueueMsg() function, TE internally notifies its receiving components, which could be blocked at a *receive.* So, Apache HTTP GET

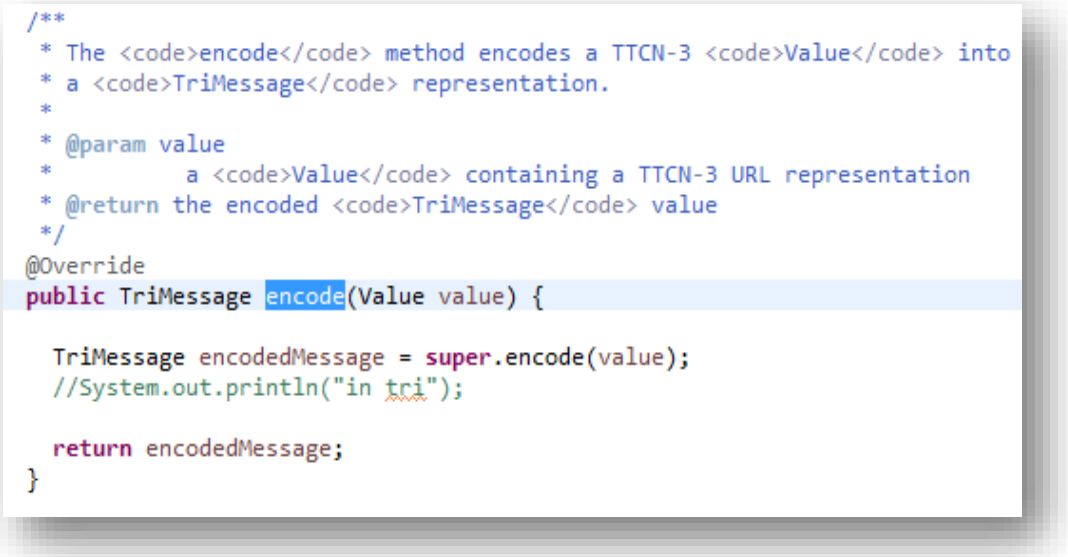
method is implemented in System Adapter to get the payload from SUT to Test Components using TTCN-3 Runtime Interface.

7.1.4 Developed Codec

In other to receive/transmit the actual data over a TTCN-3 port to/from the SUT, we must require few entities like encoder and decoders. Encoders can encode the transmitting data to SUT representation. On the other hand, approaching information also need to be read. Decoders are used to decode the incoming data from SUT, to allow coordinating and matching against TTCN-3 template definitions for quality assurance.

Codecs are small program responsible to encode and decode the values of TTCN-3 types understandable to SUT. These are specified in the standard TTCN-3 Control Interface (TCI-CD) (TTCN-3, 2014). It is feasible if one wants to write their own encoders and decoders. Decoder contains the decodenodes() function which could tell the codec which all types are present in the test case, and then it should assign a codec for each type and values. This entity basically responsible for providing TCI-Codecs, which are used to encode TTCN-3 values into transmission syntax form (if any data is sent to SUT) and decode data in transmission syntax form back to TTCN-3 understandable format (decoders comes under the picture in the case whenever we are receiving the data from SUT). Usually, we write single system for the codec which include encoders and decoders using the functions encode() and decode().

In below figure 7-10 you can see the code representation of encode() function, which is used to encrypt the TTCN-3 values into TriMessage representation to send it further to SUT.



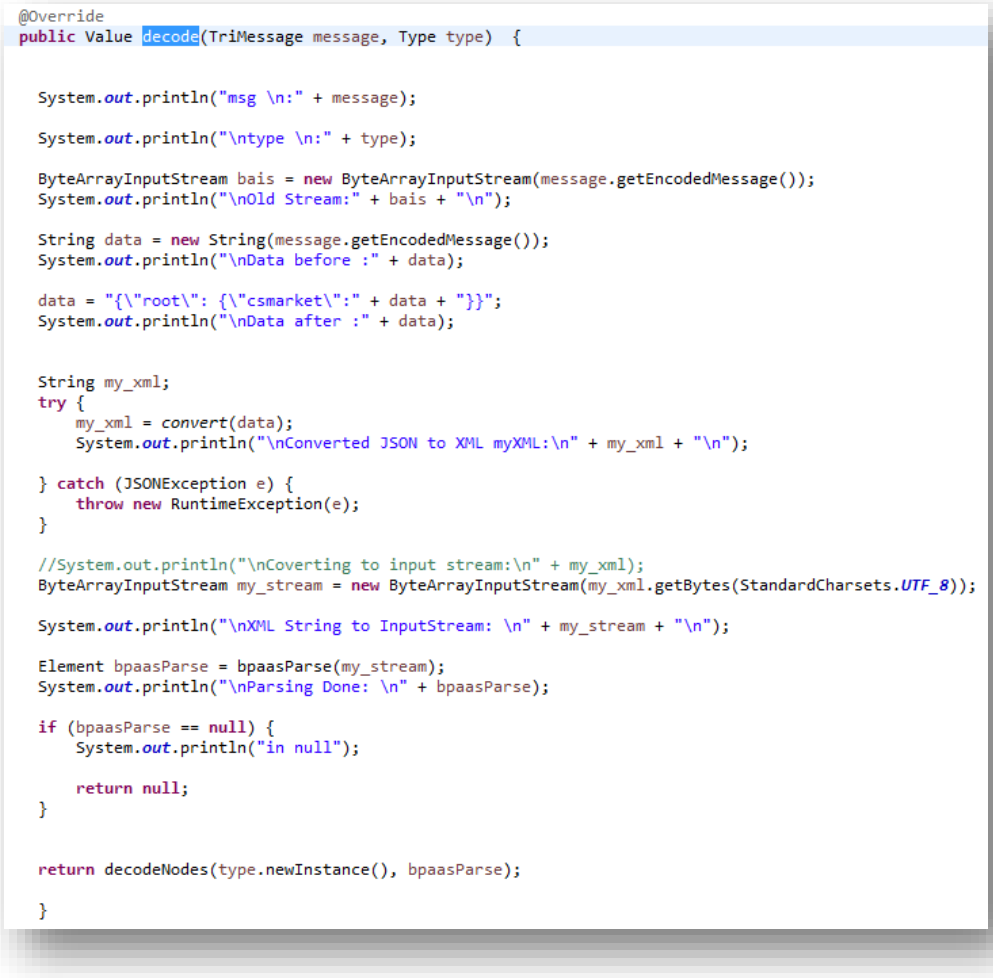
```
/**
 * The <code>encode</code> method encodes a TTCN-3 <code>Value</code> into
 * a <code>TriMessage</code> representation.
 *
 * @param value
 *          a <code>Value</code> containing a TTCN-3 URL representation
 * @return the encoded <code>TriMessage</code> value
 */
@Override
public TriMessage encode(Value value) {

  TriMessage encodedMessage = super.encode(value);
  //System.out.println("in tri");

  return encodedMessage;
}
```

Figure 7-10: TCI-CD encode() function representation

On the other hand, decode () function is responsible to read and understand the data receiving from SUT. It converts the Bitstring data to TTCN-3 data representation. I wrote decoder according to the requirement of my case study so that our data can be matched to the expect data to assess the quality of the service over Cloud. This method basically decodes the TTCN-3 Value receiving from SUT in form of TriMessage representation. In this case, TriMessage contains JSON structured payload. TE may query more than one time for the same bitstring to parse the data. In below code snippet Figure 7-11, decoding process is explained that how data is streamed into bytearrayinputstream().



```
@Override
public Value decode(TriMessage message, Type type)  {


  System.out.println("msg \n:" + message);

  System.out.println("\ntype \n:" + type);

  ByteArrayInputStream bais = new ByteArrayInputStream(message.getEncodedMessage());
  System.out.println("\nOld Stream:" + bais + "\n");

  String data = new String(message.getEncodedMessage());
  System.out.println("\nData before :" + data);

  data = "{\"root\": {\"csmarket\":" + data + "}}";
  System.out.println("\nData after :" + data);


  String my_xml;
  try {
      my_xml = convert(data);
      System.out.println("\nConverted JSON to XML myXML:\n" + my_xml + "\n");

  } catch (JSONException e) {
      throw new RuntimeException(e);
  }

  //System.out.println("\nCoverting to input stream:\n" + my_xml);
  ByteArrayInputStream my_stream = new ByteArrayInputStream(my_xml.getBytes(StandardCharsets.UTF_8));

  System.out.println("\nXML String to InputStream: \n" + my_stream + "\n");

  Element bpaasParse = bpaasParse(my_stream);
  System.out.println("\nParsing Done: \n" + bpaasParse);

  if (bpaasParse == null) {
      System.out.println("in null");

      return null;
  }


  return decodeNodes(type.newInstance(), bpaasParse);

  }
```

Figure 7-11: TCI-CD decode() function representation

There could be many System.out.println() commands used in the source code, which is solely used for debugging purpose. This above decode() function is designed to handle both JSON and XML type of payload structure. Since our payload from *csmarket* servers is in JSON format, hence we will use convert() function to convert the received payload into XML ByteArrayInputStream, which is further parsed elements by element to check and match the quality of the data received as per expectation. The parser is responsible about the parsing of the function parameter InputStream input. It will throw the TciException if an error occurs during the parsing.

```
/**
 * Cares about the parsing of the InputStream input.
 *
 * @param input
 *          an <code>InputStream</code> to parse
 * @return the document root <code>Element</code>
 * @exception TciException
 *              if an error occurs during parsing
 */
private Element bpaasParse(InputStream input) // throws TciException
{

  if (getRB().debug) {
    logDebug("Codec: parse(InputStream input) entered");
  }
  try {
    DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
    docBuilderFactory.setValidating(false);

    DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
    Document document = docBuilder.parse(input);
    input.close();

    return document.getDocumentElement();
  }
  catch (Exception e) {
    setDecodingError("Failed to parse input: " + e.getMessage(), null);
  }

  return null;
}
```

Figure 7-12: TCI-CD Representation of parser() function used by decode()

Figure 7-13: Represents convert() to be used by decode()

```
public static String convert(String json) throws JSONException
{
    org.json.JSONObject jsonStringObject = new org.json.JSONObject(json);
    String xml = "<?xml version=\"1.0\" encoding=\"ISO-8859-15\"?>" + org.json.XML.toString(jsonStringObject);
    return xml;
}
```

Convert() is used by decoder; in case if it JSON payload is received convert() function will be called to convert the structure of the JSON to XML so that it can be parsed into more structured way.

```
private Value decodeNodes(Value value2feed, Node node) {

  Type type = value2feed.getType();

  switch (type.getTypeClass()) {

  case TciTypeClass.RECORD:
  case TciTypeClass.SET:
      System.out.println("\nIn case 1 of type: " + type);

    if (node != null && node.getNodeType() == Node.ELEMENT_NODE) {
      RecordValue record = (RecordValue) type.newInstance();
      String[] ttcnFieldNames = record.getFieldNames();
      NodeList nodeChildren = node.getChildNodes();

      for (String element : ttcnFieldNames) {
        Value fieldValue = record.getField(element);

        for (int i = 0; i < nodeChildren.getLength(); i++) {
          Node child = nodeChildren.item(i);

          switch (fieldValue.getType().getTypeClass()) {
          case TciTypeClass.RECORD:
          case TciTypeClass.RECORD_OF:
          case TciTypeClass.SET:
          case TciTypeClass.SET_OF:
          case TciTypeClass.CHARSTRING:
          case TciTypeClass.INTEGER:

            if (child.getNodeType() == Node.ELEMENT_NODE
            && child.getNodeName().equals(element)) {
              record.setField(element, decodeNodes(fieldValue, child));
            }

            break;

          default:
            tciErrorReq("Unsuppported Type " + fieldValue.getType().getName());
```

Figure 7-14: Source code snippet for representation of decodenode() function

Other function used in TCI-CD Codec is decodeNodes(). It decodes the XML/JSON code into a TTCN-3 Value representation by matching TTCN-3 templates Types and Values to the structure mentioned in this function. This method is usually invoked recursively to decode structured payload. Function parameter value2feed are the expected

TTCN-3 Value values, and node parameter in the function is pointing towards the payload structure nodes. This function will return decoded TTCN3 Value to the function decode().

7.2 CASE STUDY AND RESULTS

In this section, we will represent the quality assurance aspects and test results of our proposed framework for testing the business processes over the Cloud. Organizations are aggressively taking their business processes online using BPM tools and technologies, which might put them into the position of an incubator to host the business process as a service, and providing support for BPaaS. And mostly, quality assurance in such scenarios where companies are offering business process as services are handled as just another testing of a web service or an application. You have already gone through with the framework for testing the business process as services in Section 7.1. In this section, you can see the detailed explanation of TTCN-3 implementations of the former. We have already explained about our SUT (i.e. CloudSocket marketplace providing Business processes as a Service).

For the business process (shown in figure 6-3), TTCN-3 was used to verify and validate the quality of the data which is received from *SUT csmarket* servers through system adapter and codecs. Firstly, we will receive some data from the server using Apache HTTP GET method in system adapter, which is decoded via code. That message from the server is presented in Test data view as shown in figure 7-15.

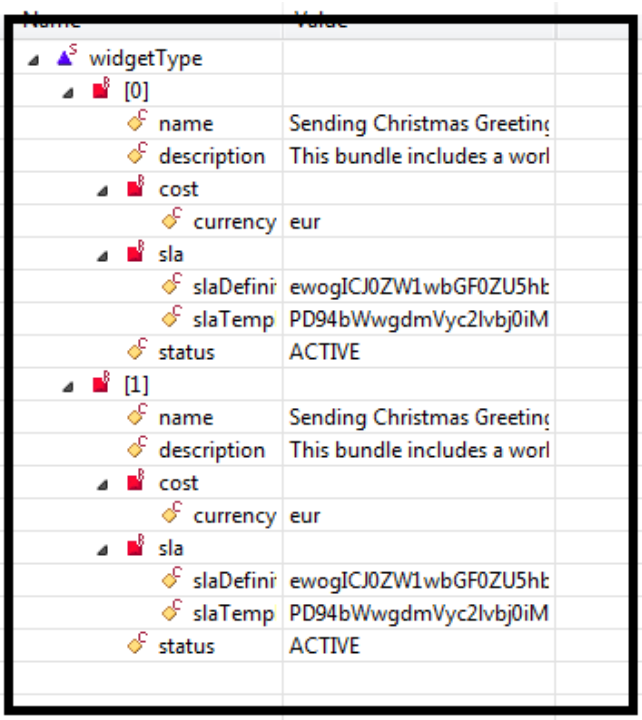


Figure 7-15: Displaying actual information received from csmarket server

Then, this received data will be matched with the data entered in TTCN-3 Template for quality assurance. Next figure 7-16 shows the template data which has to be matched for the quality assessment.

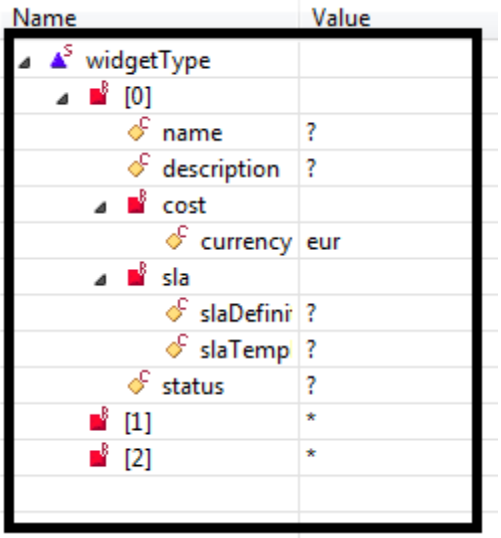


Figure 7-16: Template for the expected message received

For example, information for different parameters of the data and uploaded image is received from SUT JSON payload and only a few are matched with the template of TTCN-3 code, to assure the quality of the Business process.

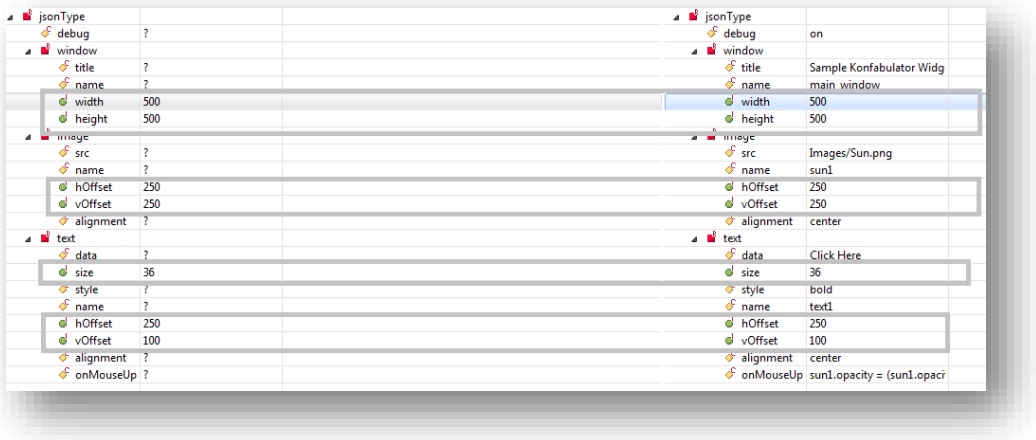


Figure 7-17: Test execution view depicting matched test parameters

TTCN-3 has a very interesting and influential characteristic of displaying the test case execution graphically; also with good reporting attribute for summarizing and concluding the test results. The whole execution of passed test is graphically presented in figure 7-18, where only 1 PTC is taken into consideration. The user can add as many as test components he/she wants to add.

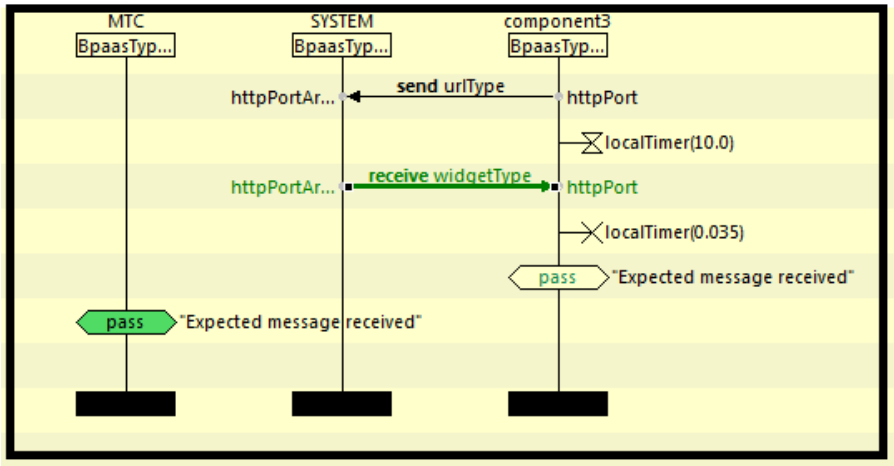


Figure 7-18: Graphical representation of passed test cases

If in above figure 7-17, if any of the test cases failed to match the quality standard mentioned in TTCN template, the whole test will be considered fail and will be logged in the report on the same time. A screenshot for the deliberately failed test case is depicted in image below, where System under test send some information to test *component3* (as seen in Figure 7-19), but it failed to match the values mentioned in the template.

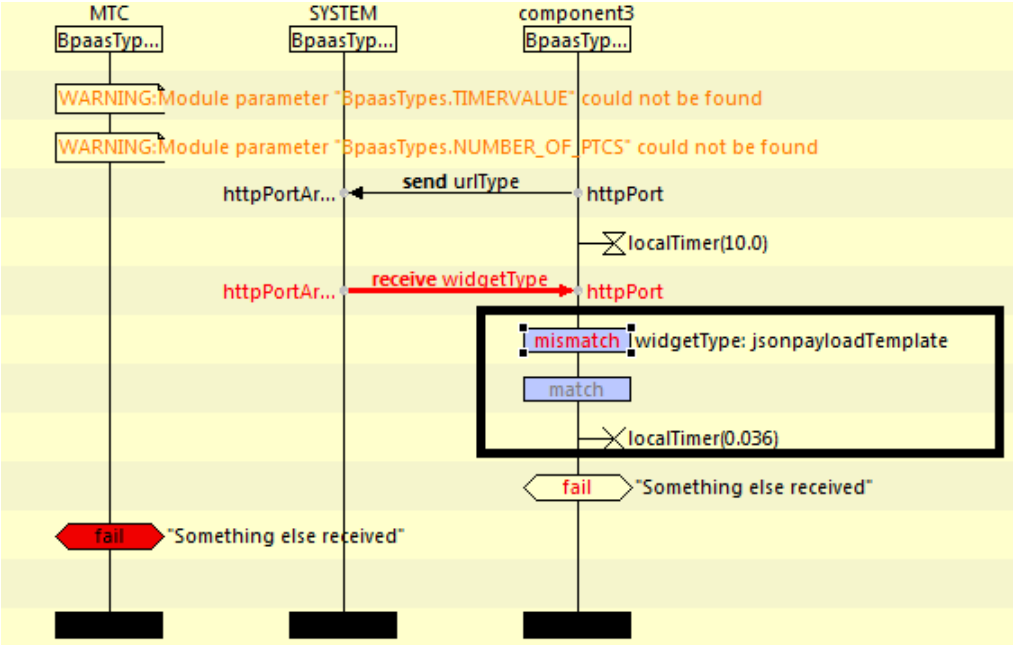


Figure 7-19: Graphical representation of failed test cases

7.3 SUITABILITY OF TTCN-3

This section discusses the suitability of TTCN-3 testing standards for testing the business processes in the Cloud-based environment. The objective of the thesis was to develop a test system to test the quality of published BPaaS bundle on the CloudSocket market before end-users can get access to them. The aim was to deploy TTCN-3 methodologies to test the BPaaS bundles with the desired quality from CloudSocket Marketplace and check if this approach is feasible to test the BPaaS bundles in a heterogeneous Cloud environment.

According to the work done in the thesis, we have found that the approach for using TTCN-3 for testing the quality of BPaaS is feasible as of current status of the project. It might get any complications as project advances to the next level. In this thesis, designed test system have three building blocks, viz. TTCN-3 Test Framework; System adapter; and Codec. TTCN-3 test framework sets the protocols for data to be matched using TTCN-3 templates. System adapter is an essential component used to build an interface between CloudSocket Marketplace (BPaaS under test) and test component (TTCN-3 test framework). It acts as a local client application designed to use Apache HTTP GET method and access the JSON payload from csmarket servers using RESTful web-services. The reason for using this approach is that we have data on the CloudSocket servers which cannot be changed. Due to that, we ought to use only GET method, as other HTTP methods are not required in this CloudSocket module. This test system is only extracting the JSON payload to match the quality of the received data. This is the reason we can conclude that, technically, it is meaningful to exploit the BPaaS CloudSocket system modeling effort towards a TTCN-3 test framework.

Our TTCN-3 testcase basically describes the extraction of the test data from the system under test (SUT) using RESTful services and HTTP get method. Adjacently codec is deployed to decode the received content and parse it to the system understandable format. In parallel, development of other essential TTCN-3 components is done using TTWorkbench. TTCN-3 templates are created which contains the expected data to be received from the web server as an answer to assess the quality of the payload. After defining the structure of the expected messages in templates, we created the test component instances in a file testcase.ttcn3. There can be a multiple number of test components, parallel test components (PTCs) which are responsible for sending the

request to CloudSocket server. Bpaasfunction.ttcn3 file is also generated which is responsible to define function ptcbehaviour(). This function is responsible for the behavior of parallel test component (PTC) against received content and provides the verdict using setverdict function. To compute the verdict Testcase for matching both the data is also written using TTCN-3.

There are the research papers based on the suitability of TTCN-3 for BPT (Mallur, 2015). The evaluation done by them is that testing business processes using TTCN-3 is not desirable and efficient approach. However they were more involved in the unit testing of business process and providing a performance reports. Resulting, they opted IBM BPM testing asset over TTCN-3 because they are not dealing with the Cloud-based environment. The main reason that TTCN-3 is used in our report is that in Cloud-based business process scenarios we are extracting the data of BPaaS bundles directly from *CloudSocket Maketplace* to test the quality of the BPaaS system. For this, we must establish the reliable message exchange interface between the test system and SUT. TTCN-3 provide this functionality in their core component TRI using *triCommunication.* It should be noted that it is possible to test the quality of the business processes provided as a service over Cloud environment using TTCN-3.

The next step of the research is to assess this testing system with even more complex scenarios to ensure the robustness of the module. To do this, we require various kinds of BPaaS bundles from CloudSocket Marketplace with different and unique requirements. More test parameters can be added to increase the quality assessment standards. This also aims to increase the TTCN-3 support for testing the BPaaS and making the test system more generic. I.e. to expand the support of the

test system with various kind of payloads. This will include development of a whole new generic codec for parsing the data and testing the BPaaS.

Another future work is to expand the TTCN-3 support for model-based testing towards the *automatic* derivation of test models together with the system model (Kavya Mallur, September 2015). However this is still ongoing work in companies associated with the CloudSocket project. The reason for the potential expansion of the CloudSocket project to support model-based testing is that apart from assessing the quality of the pre-designed business process, the project in principle needs the testing of all stages of the BPaaS lifecycle. Model-based testing then would exploit eventually both the methodologies of the *BPaaS design environment* (the BP level) and of the *allocation environment* (the Cloud level) (meta-modelling and semantic annotations). Such a TTCN-3 oriented testing frameworks would represent a key element of a “Quality Assessment Framework for Business Process as a Service in a heterogeneous Cloud Environment”.

8 CONCLUSION

In this thesis, we have introduced an initial work for testing Business Processes as a Service in a heterogeneous Cloud Environment. We have taken few steps to get to the conclusion of this thesis which includes; (a) State of the Art for Business process testing, (b) The selection of Decision table testing for the testing of Business processes, (c) State of the Art for Testing the Cloud, (d) Understanding the concept of Cloud testing and introducing the method BPaaS – where business processes are provided as a service over the Cloud, (e) Implementation of test cases in TTCN-3 and this initial test case was implemented and validated against the CloudSocket marketplace. (f) Finally we have commented on the suitability of TTCN-3 within a "Quality Assessment Framework for Business Process as a Service in a heterogeneous Cloud Environment".

9 REFERENCE

The NIST Definition of Cloud Computing. (2011, October 24). *Final Version of NIST Cloud Computing Definition Published*. Retrieved from NIST: https://www.nist.gov/news-events/news/2011/10/final-version-nist-cloud-computing-definition-published

Apache HTTP Project. (2005-2016). Retrieved from Apache HTTP: https://hc.apache.org/httpcomponents-client-ga/examples.html

Apprenda Inc. (2016). *IaaS, PaaS, SaaS (Explained and Compared)*. Retrieved from APPRENDA: https://apprenda.com/library/paas/iaas-paas-saas-explained-compared/

Bertram, A. (2015, July 10). *How To Build An IaaS Test Lab In Windows Azure*. Retrieved from tom's IT PRO: http://www.tomsitpro.com/articles/windows-azure-iaas-test-lab,2-699.html

Cai Ferriday, T. D. (2007). *A Review Paper on Decision Table-Based Testing.*

Cloud Standards Customer Council. (2015). *Practical Guide to Platform-as-a-Service.* Cloud Standards Customer Council.

CloudSocket. (2015). *CloudSocket common understandings wiki*. Retrieved from cloudsocket.eu: https://www.cloudsocket.eu/web/guest/common-understanding-wiki

CloudSocket. (2016). *Demonstration: End-User Perspective*. Retrieved from CloudSocket: https://www.cloudsocket.eu/web/guest/demonstration-end-user-perspective

CloudSocket Consortium. (2015, August 31). FIRST CLOUDSOCKET ARCHITECTURE. *CloudSocket*. CloudSocket Consortium.

CloudSocket. (European Union's Horizon 2020 Framework Programme). *CloudSocket Idea*. Retrieved from CloudSocket: https://www.cloudsocket.eu/project

Danilo Ardagna, G. C. (2014). Quality-of-service in cloud computing: modeling techniques and their applications. In G. C. Danilo Ardagna, *Journal of Internet Services and Applications.* BioMed Central Ltd.

Dr. Rahul Malhotra, P. J. (2013). *Testing Techniques and its Challenges in a Cloud Computing Envirnment.* Punjab, India: The Standard International Journals (The SIJ) .

Frank Leymann, D. R. (2000). *Production Workflow: Conceepts and techniques.* Presntice Hall.

Gilad, Z. (2011, October 19). *The Problem with Quality in the Cloud.* Retrieved from Database rends and application: http://www.dbta.com/Editorial/Think-About-It/The-Problem-with-Quality-in-the-Cloud-78187.aspx

Grance, P. M. (2009, July 10). *The NIST Definition of Cloud Computing*. Retrieved from National Institute of Standards and Technology, Information Technology Laboratory: https://www.nist.gov/sites/default/files/documents/itl/cloud/cloud-def-v15.pdf

HYRKKÄNEN, A. (2004, June 9). *General Purpose SUT Adapter for TTCN-3*. Retrieved from TTCN-3: http://www.ttcn-3.org/files/GeneralPurposeTTCN3SA.pdf

itSMF UK. (2012). *An Introductory Overview .* London: TSO (The Stationery Office) .

Kavya Mallur, M. A. (September 2015). A Model-based Quality Assurance Framework for Online Business Processes. *ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems* (p. 4). Ottawa, Ontario, Canada: ResearchGate.

Kees Blokland, J. M. (2013). *Testing Cloud Services.* California: Polteq.

Mallur, K. (2015, March 30). A Quality Assurance Framework for Business Process Management. University of Ottawa, Ontario, Canada.

McNickle, M. (2012, February 2). *5 issues affecting cloud service quality and performance.* Retrieved from http://www.healthcareitnews.com/news/5-issues-affecting-cloud-service-quality-and-performance

Mengerink, P. B. (2013). *Testing Cloud Services, How to Test Saas, Paas & Iaas.* ISBN 978-1-937538-38-5: Rockynook Inc.

Parastoo Mohagheghi, J. A. (2007). Evaluating Quality in Model-Driven Engineering . *29th International Conference on Software Engineering Workshops(ICSEW'07* (p. 2). Oslo, Norway: IEEE.

Rouse, M. (2016, May). *Software as a Service (SaaS).* Retrieved from Tech Target: http://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service

Sogeti GmbH, Capgemini. (2014, June 26). *Testing Platform-as-a-service*. Retrieved from Sogeti: https://www.sogeti.com/solutions/testing/specialized-testing-services/testing-platform-as-a-service/

Test strategy, Testing Methodologies. (2016, December 14). *How to Write Complex Business Logic Test Scenarios Using Decision Table Technique*. Retrieved from Spftware Testing Help: http://www.softwaretestinghelp.com/decision-table-test-case-design-technique/

Testing Concepts, T. M. (2016, December 14). *Business Process Testing (BPT) – How to Simplify and Speed Up the Testing Process Using BPT*. Retrieved from Software testing help:

http://www.softwaretestinghelp.com/what-is-business-process-testing-bpt/

Testing Concepts, Testing Methodologies. (2016, December 14). *Business Process Testing (BPT) – How to Simplify and Speed Up the Testing Process Using BPT*. Retrieved from Software testing help: http://www.softwaretestinghelp.com/what-is-business-process-testing-bpt/

Testing Technologies IST GmbH. (23 November 2015). *Testing Technologies TTworkbench User's Guide.* Testing Technologies IST GmbH.

Testing_Tech. (2016). *The execution of TTCN-3 tests: The TTCN-3 Runtime and Control Interfaces.* www.testingtech.com.

Thomas Barton, C. S. (2014). *BusinessProcessasa Service– Status and architecture.* University of Applied Sciences Worms, University of Applied Sciences Landshut.

TMap Sogeti. (2014, October 24). *Process Cycle Test (PCT).* Retrieved from www.tmap.net: http://www.tmap.net/wiki/process-cycle-test-pct

TTCN-3. (2013). *Application Domains*. Retrieved from TTCN-3 : http://www.ttcn-3.org/index.php/about/references/applicatio-domains

TTCN-3. (2014, December 09). *Introduction*. Retrieved from TTCN-3: http://www.ttcn-3.org

TTCN-3. (2014, December 09). *TTCN-3 Test System Reference Architecture*. Retrieved from TTCN-3: http://www.ttcn-3.org/index.php/about/referrence-architecture

TTWorkbench - Spirent. (2016, November). *TTworkbench: Build, Execute, and Analyze Complex Test Scenarios*. Retrieved from Spirent Commuications Inc.: https://www.spirent.com/-/media/Datasheets/TT/TTworkbench.pdf?la=en

Worksoft. (2015, July 20). *Lights Out Business Process Testing. What it is. Why you need it.* Retrieved from Worksoft Inc.: https://www.worksoft.com/files/resources/Lights-Out-Business-Process-Testing.pdf