And MBTAAS for IoT Platforms

Amjad Mahfoud

IoT Mashups:

Mashup tools have been proposed as a simple way to develop applications by ***composing***, or ***mashing-up***, existing services in the Web.

This was supported by increasingly uniform communication protocols and APIs based on REST principles[3].

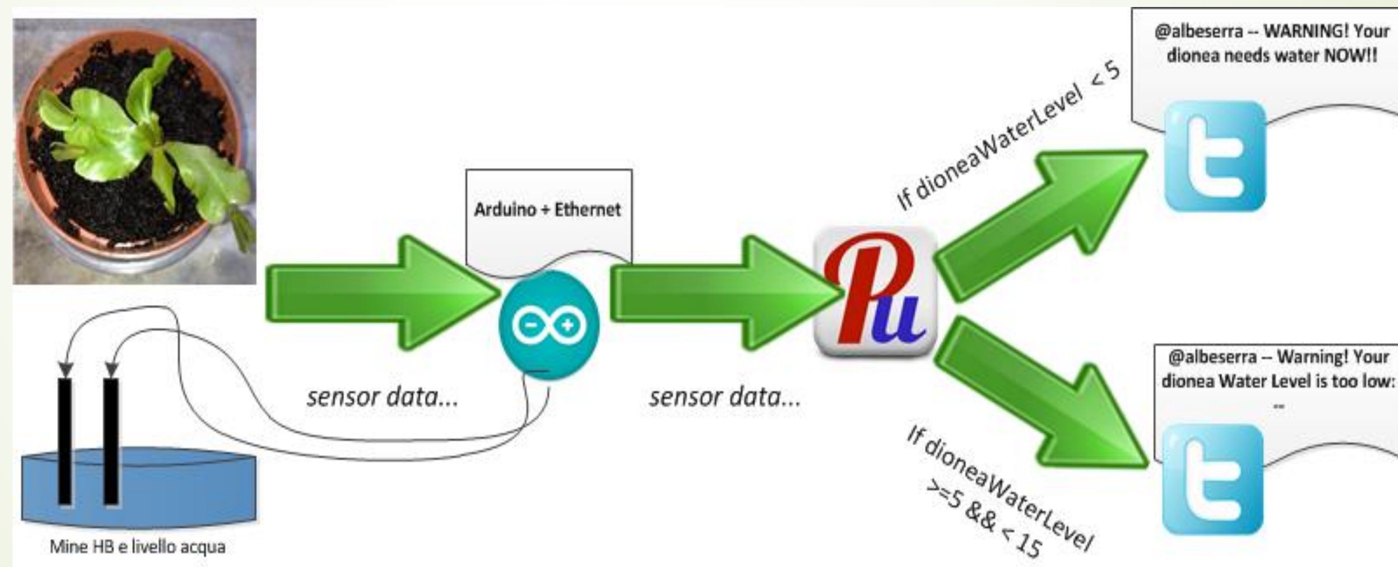
Early mashup tools are Microsoft Popfly and Yahoo Pipes.

In the recent years, there has been a lot of interest in applying the same ideas to the Internet of Things, also building on REST interfaces for the Internet of things [3].

Well known examples for such tools are *Clickscript, WotKit as well as Paraimpu* [3].

For the connecting services, there are different concepts. The main, predominant one is ***modeling data flow***.

For others, mainly in the enterprise area, also centralized approaches are considered. As an example, we show an illustration from Paraimpu. This shows a typical flow from sensor data, to some processing up to the connection with Web-based services.



Mashup tools typically provide a graphical editor for the composition of services for one application.

This typically models the message flow between the components.

Components can be sensor nodes, processing or aggregation entities as well as external Web-based services.

Thus, mashup tools can also be seen as specific forms of end-user programming. but are however limited to the specific model of describing message flow.

In addition, some mashup tools provide simulation tools and also interoperability for messaging between different platforms.

Model-Based IoT

There is a broad range of model-based approached, up to domain specific modeling languages.

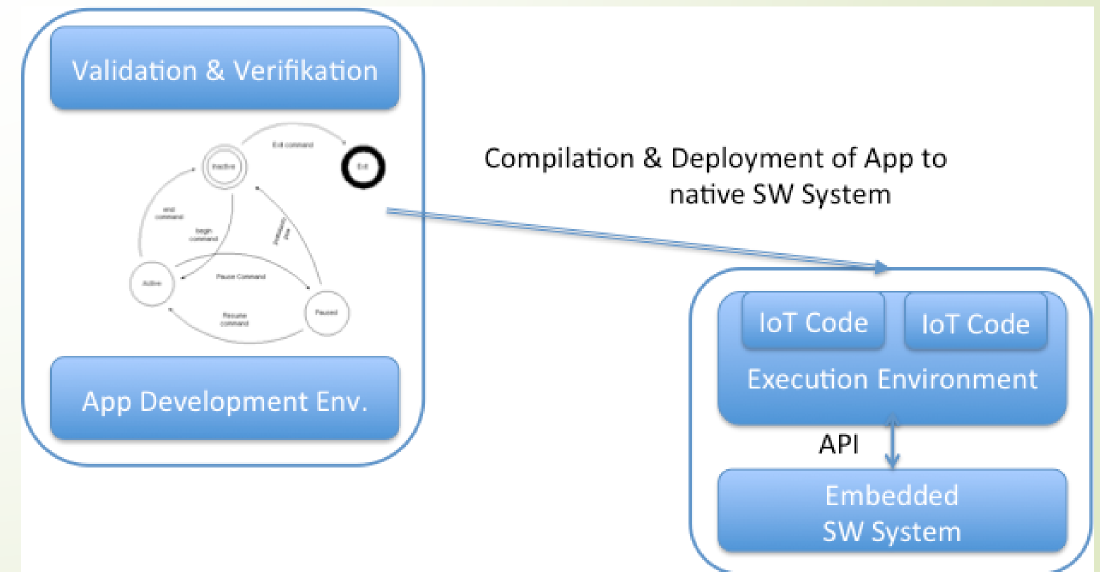
Here, we mainly assume general purpose modeling tools like UML, even though many more specific approaches exits.

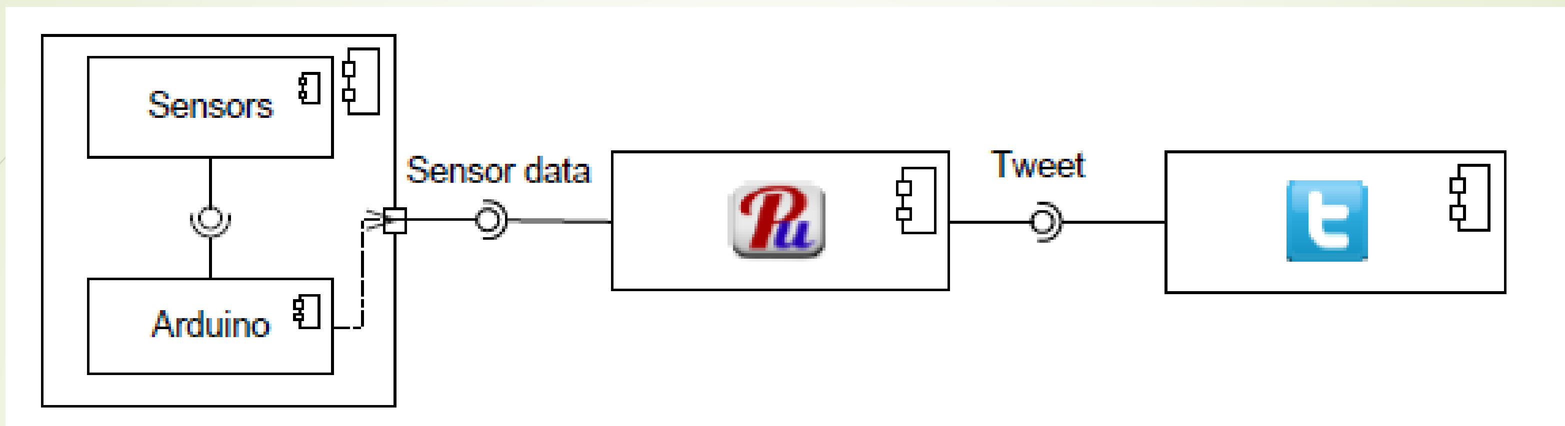
For instance, there are several proposals for model-based approaches for developing IoT applications, e.g. the *ThingML* language as well as other proposals [3].

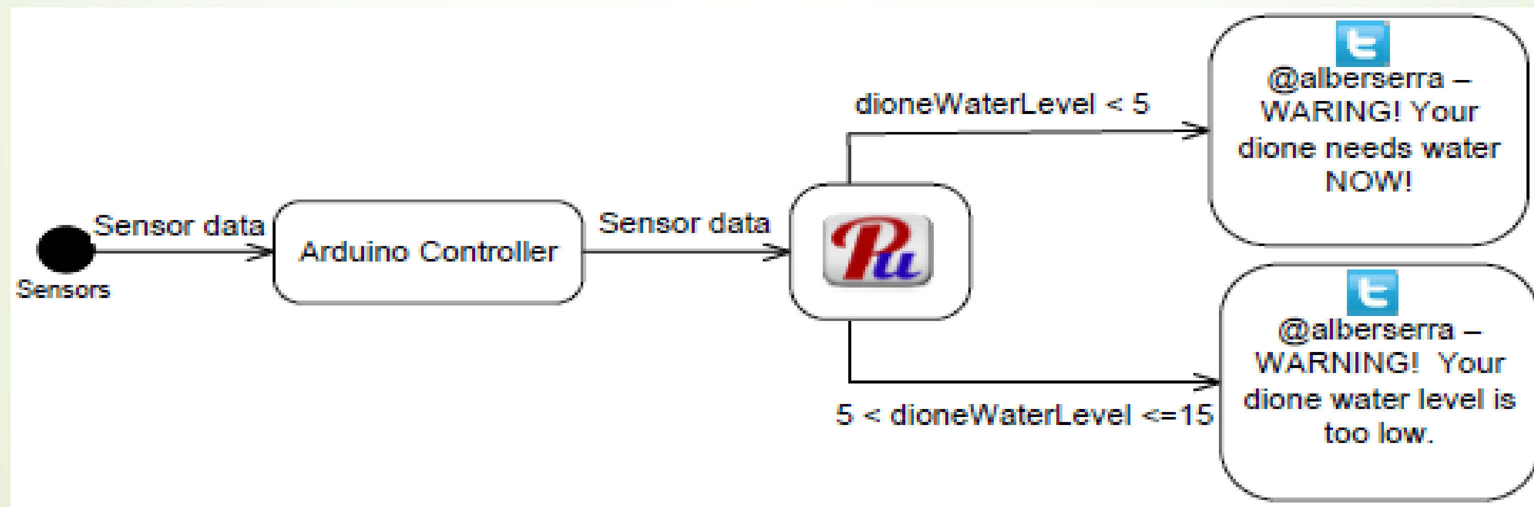
The motivation for model-based development is *to describe a system on a higher level of abstraction*. Typically, this is done in UML and other languages by diagrams modeling specific aspects or views of a system.

Behavior can be described by examples in sequence diagrams, or by state machines and activity diagrams. ***Activity diagrams describe the data and event flow, similar to mashup tools***.

State diagrams are used in many embedded domains to model the behavior of specific objects. Also, state diagrams can be analyzed and verified formally and code can be generated automatically.



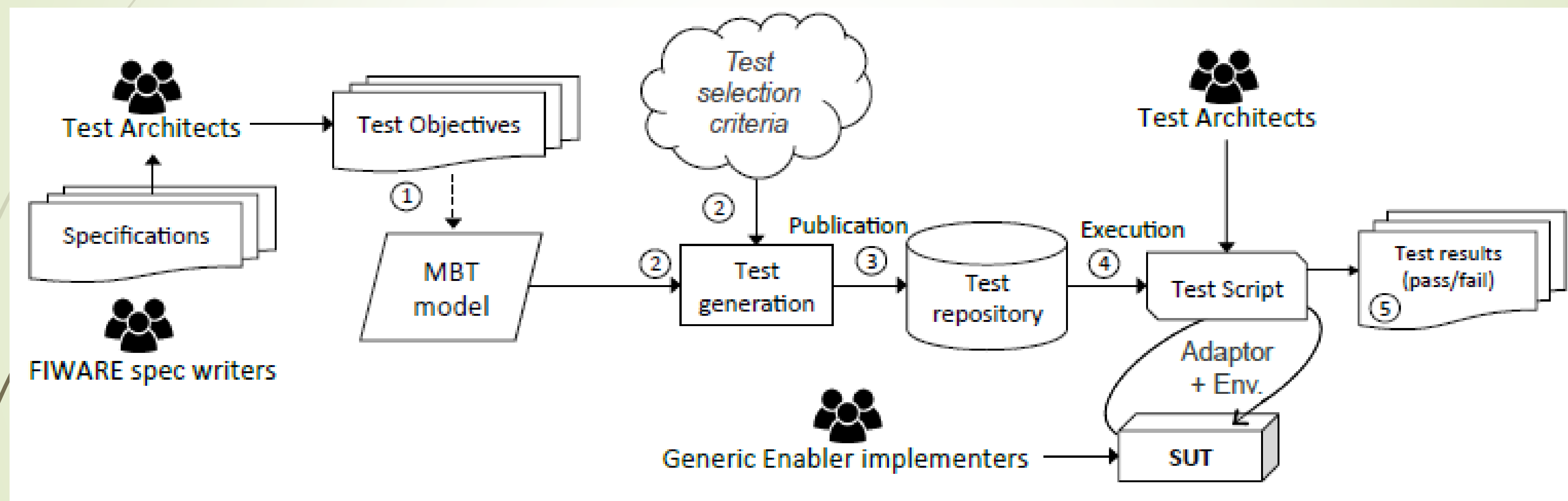

Sensors
Arduino
Sensor data
Tweet


Sensors
Sensor data
Arduino Controller
Sensor data
dioneWaterLevel < 5
@alberserra – WARING! Your dione needs water NOW!
5 < dioneWaterLevel <=15
@alberserra – WARNING! Your dione water level is too low.

MBT in IoT

Indeed, most recent IoT platforms are using ***standardized protocols*** to communicate (MQTT, CoAP, HTTP).

This makes MBT testing deployment very suited by enabling design of a generic model, based on these standards and producing test cases that can be used over multiple applications.


Test Architects
Test Objectives
Specifications
FIWARE spec writers
1
MBT model
Test selection criteria
2
2
Test generation
Publication
3
Test repository
Execution
4
Test Architects
Test Script
Test results (pass/fail)
5
Adaptor + Env.
Generic Enabler implementers
SUT

MBTAAS in IoT

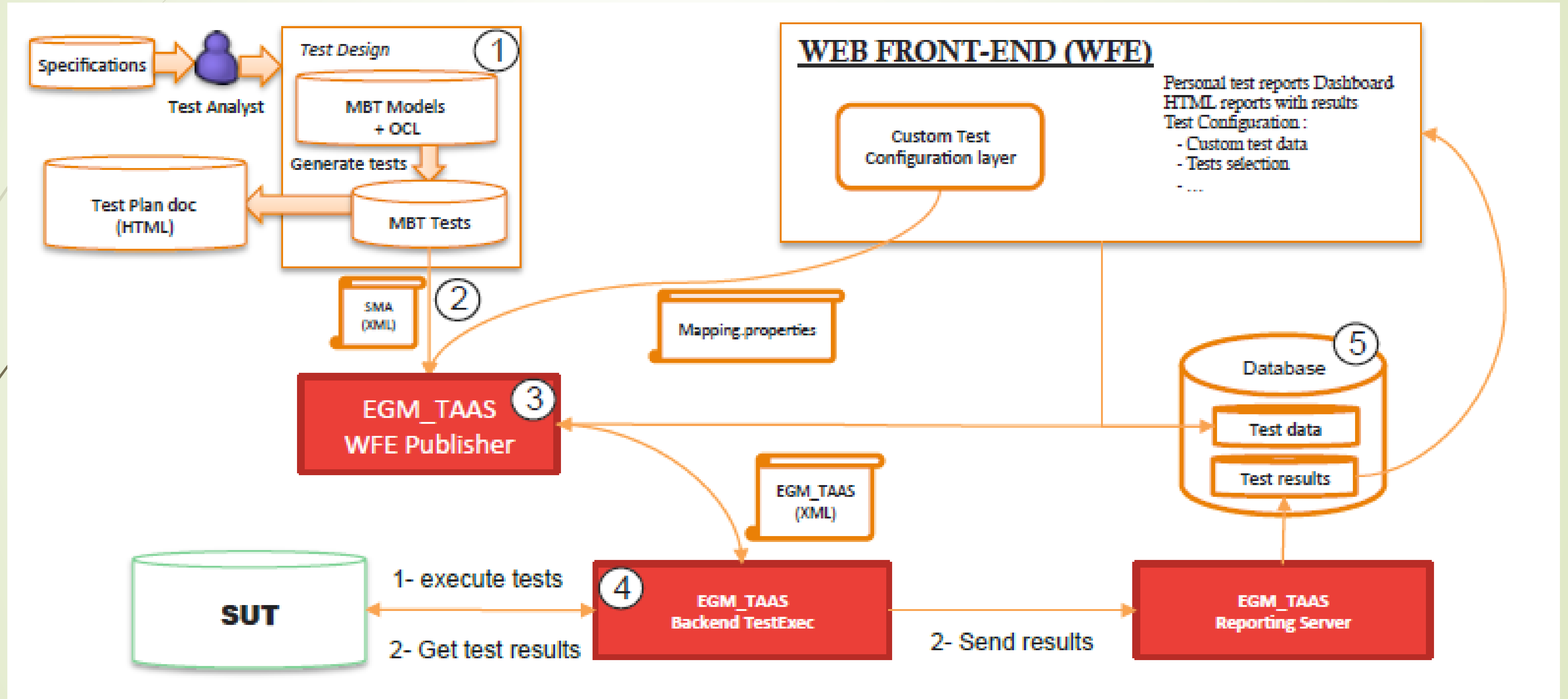
IoT platforms offer services to applications users.

The question of conformance testing and validation of IoT platforms can be tackled with the same ”as a service” approach.

This section presents the general architecture of our Model Based Testing As a Service (MBTAAS).

We then present in more details, how each service works individually in order to publish, execute and present the tests/results.

Architecture:



However, to the difference of a classical MBT process, MBTAAS implements several ***web services***, which communicate with each other in order to realize testing steps.

A web service, uses web technology such as HTTP, for machine-to-machine communication, more specifically for transferring machine ***readable*** file formats such as XML and JSON.

In addition to the classical MBT process, the central piece of the architecture is the ***database service*** that is used by all the other services

Customization Service:

In order to provide a user friendly testing as a service environment, they created a graphical web-front end service to configure and launch test campaigns.

The web-service enables:

- **Test selection:** from the proposed test cases, a user can choose to execute only a part of them.
- **Test Data:** pre-configured data are used for the tests. The user is able to add his own specific test data to the database and choose it for a test. It is a test per test configurable feature.
- **Reporting:** by default the reporting service will store the result report in the web-front end service. The default behavior can be changed to fit the user needs for example, having the results in an other database, tool, etc.

After completion of the test configuration and having the launch tests button

pressed, a configuration file is constructed. The configuration file as can be seen

Configuration File excerpt, defines a set of {key = value}.

This file is constructed with default values that can be overloaded with user defined values.

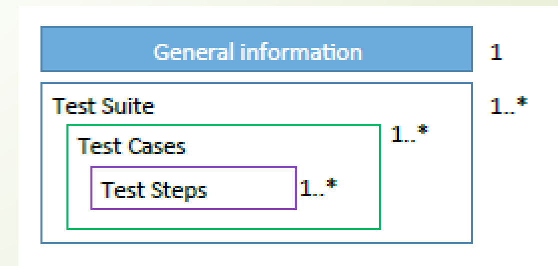
```
14  ##################################################################################
15  ###############    REQUIRED PARAMETERS   ##########################################
16  ##################################################################################
17
18  #NAME OF THE OWNER OF THE REPORT
19  OWNER=EGM_TE_XML_PUBLISHER
20  #REPORT LOCATION AFTER TESTS (FOR EGM_TAAS_BACKEND)
21  REPORT_LOCATION=http://193.48.247.210:8081/report
22  #HOW TO REPORT (FOR EGM_TAAS_BACKEND)
23  REPORT_TYPE=POST_URL
24  #URL OF SUT TO TEST WITH THE PORT (FULL PATH)
25  ENDPOINT_URL=http://193.48.247.246:1026
26  #URL of EGM_TAAS backend that will execute the tests
27  EGM_TAAS_BACKEND = localhost:8080/executeTests
28  #Name of the Model file to be Used by EGM_TAAS_BACKEND
29  EGM_TAAS_MODEL = OrionCB_GE.xml
30  #Where to Output the results in the EGM_TAAS_BACKEND
31  EGM_TAAS_OUTPUT = tmp
```

Publication service:

The publisher service, as it name states, ***publishes*** the abstract tests generated from the model into concrete test description file. It requires three different inputs for completion of its task:

1. the model,
2. the configuration file
3. test data.

The model provides test suites containing abstract tests. The concretization of abstract tests is made with the help of the database and configuration file.



Execution service:

The execution service **is the functional core** of the **MBTAAS** architecture. The execution service will run the test and collect results depending on the configuration of the received test description file.

Each test is run against the SUT and a result report (next slide) is constructed on test step completion.

```xml
<teststep name="UpdateEntity1">
  <executionResults>
      <timestamp>{TIMESTAMP}</timestamp>
    <executionTimeMs>22</executionTimeMs>
  </executionResults>
  <endpoint>
    <value>{IP}:{PORT}/upContext</value>
    <isinvalid>false</isinvalid>
  </endpoint>
  <method>POST</method>
  <headers>{HEADERS}</headers>
  <payload>{PAYLOAD}</payload>
  <assertion_list>
    <assertion>
        <type>JSON</type>
          <key>code</key>
        <value>404</value>
        <result>false</result>
      </assertion>
    </assertion_list>
    <result>false</result>
    <response>{
      "errorCode" : {
        "code" : "400",
        "reasonPhrase" : "Bad Request",
        "details" : "JSON Parse Error"
      }
    }
      </response>
  </teststep>
```

An Example of execution log

```
[egm.modelTools.HttpRequestExecuter] Validating url : http://[redacted]:1026/v1/updateContext
[egm.modelTools.HttpRequestExecuter] URL is VALID
[egm.modelTools.HttpRequestExecuter] Starting Jetty HTTP Client
[egm.modelTools.HttpRequestExecuter] Jetty HTTP Client Started with Success
[egm.modelTools.HttpRequestExecuter] Creating request : URL = http://[redacted]:1026/v1/updateContext, HTTPMETHOD = POST
[egm.modelTools.HttpRequestExecuter] Request created
[egm.modelTools.HttpRequestExecuter] Request status code: 200
[egm.modelTools.HttpRequestExecuter] Response content: {
  "errorCode" : {
    "code" : "400",
    "reasonPhrase" : "Bad Request",
    "details" : "JSON Parse Error"
  }
}

[egm.modelTools.HttpRequestExecuter] Stopping Jetty HTTP Client
[egm.modelTools.HttpRequestExecuter] Jetty HTTP Client Stopped
[egm.modelTools.HttpRequestExecuter]

[egm.model.Assertion] Asserting...expression to be assert: is key "code" contains value: "404"
[egm.modelTools.TestStepResponseParser] is JSON data key "code" contains value: "404"
[egm.modelTools.TestStepResponseParser] no value of : "404" has been found for id: "errorCode"
[egm.modelTools.TestStepResponseParser] no value of : "404" has been found for id: "code"
[egm.model.TestStep] Execution Result of step UpdateEntity148 : false
```

Reporting service:

After executing the test, a file containing the test description alongside their results are sent to and received by the reporting service.

The reporting service configuration is made in the web front-end service.

The configuration is passed with the test configuration file where the publisher service re-transcribes that information to the file sent to the execution service.

The execution service then includes the reporting configuration in the report file where it is used in the reporting service once it receives it. By default the reporting service will save the results in the database service.

Questions?

References:

References:

1. http://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT [Online; accessed 15-april-2017]
2. Gubbi, Jayavardhana, et al. "Internet of Things (IoT): A vision, architectural elements, and future directions." *Future generation computer systems* 29.7 (2013): 1645-1660.
3. Prehofer, Christian, and Luca Chiarabini. "From IoT Mashups to Model-based IoT."
4. Ahmad, Abbas, et al. "Model-Based Testing as a Service for IoT Platforms." *International Symposium on Leveraging Applications of Formal Methods*. Springer International Publishing, 2016.
5. The FIWARE Project. https://www.fiware.org/developers-entrepreneurs/, [Online; accessed 15-april-2017]