TaaS – Test as a Service

Anand Bagmar

[abagmar@gmail.com](mailto:abagmar@gmail.com)

Blog: <http://essenceoftesting.blogspot.in/>

Twitter: @BagmarAnand

[TaaS](https://github.com/anandbagmar/taas) is available on github (<https://github.com/anandbagmar/taas>)

Enterprise-sized organizations have multiple products under their belt. The technology stack used for each of the product is usually different – for various reasons.

Most of such organizations like to have a common Test Automation solution across these products in an effort to standardize the test automation framework.

**However, this is not a good idea! If products in the same organization can be built using different / varied technology stack, then why should you pose this restriction on the Test Automation environment?**

**Each product should be tested using the tools and technologies that are “right” for it.**

“**TaaS - Test as a Service**” is a product that allows you do achieve the “correct” way of doing Test Automation.

# The problem

Lets try to understand the problem in form of a simple case study.

## Case Study

Lets consider an email solution provider – that uses various different flavors of Outlook to ensure a wide coverage for its user-base, as depicted in the picture below.

NOTE: I am making some assumptions about the technical stack used for this product. This is an “almost-hypothetical” case study.

If we look at the architecture of each of the products, we will observe the following:

1. Outlook for Windows OS is developed using .Net technologies and uses a Microsoft SQL Server as its database.
2. Outlook for Mac OS is developed using Objective C on iOS
3. Outlook Web Access is developed using ASP.Net
4. Outlook Sync on Android is developed as a native Android application.

Clearly, given such a wide variety of technologies used, when it comes to Test Automation, there is no “one” tool or technology that be used to do Test Automation across **all** the products.

In fact, if you do proper analysis, you may come up with the following set of tools and technologies to use for automating each of the products referenced in our Case Study.

Now that we have understood the product and the test automation technical stack, we will realize that we have missed one core aspect of testing.

## End-2-End integration test:

For a user, who has:

* installed the desktop version of Outlook on Windows OS and on Mac OS,
* installed the Outlook native app on Android, and
* setup his / her email account in Outlook on all the above mentioned platforms

How will you automate this test:

### “An email drafted in one product should be reflected in the other products”

This test can be broken down as follows:

* Create an email in Outlook on Windows OS and save it as draft
* This draft email should be seen on Outlook on the Mac machine
* This draft email should be seen in Outlook Web Access (OWA) in a browser
* This draft email should be seen on Outlook Sync on the Android device
* The user changes the draft email content from OWA
* The updated content should be seen in Outlook on Windows OS
* The updated content should be seen in Outlook on Mac OS
* The updated content should be seen in Outlook on Android device
* …
* …

## So, how will you automate such an integration test?

# The solution – “TaaS”

## What is Taas?

TaaS is a product that will allow interaction between independent systems. The architecture is inspired by SOA.

## Architecture

End-2-End Integration Test Framework

TaaS Client

Test Framework for

Outlook on Windows

TaaS Server

Test Framework for Outlook Sync on Android

TaaS Server

**1**

**4**

**2**

3

Orchestrator

There are 2 components to TaaS:

1. TaaS Server
2. TaaS Client

## TaaS Server

This is the system that “provides service” to whoever needs to get something done on the product / test framework the TaaS server is hooked into.

TaaS Server is implemented as a REST service.

Here are the steps you need to follow for the server side:

1. Define the contract in the taas\_contracts.yml file
2. Implement the contract in your framework
3. Start the Sinatra service which will expose the contract to your end-users as a web service implementation

## TaaS Client

This is the system that “needs” to get something done in another system, without really wanting to know how it was done.

The other system could be the actual product you are interacting with, or a test framework.

Here are the steps you need to follow on the client side:

1. Browse the contracts available for consumption (as provided by the TaaS server)
2. Based on the contract you need to invoke:
   1. create a request with correct parameter values,
   2. invoke the contract,
   3. parse the output,
   4. continue your test