Crazy title

Murat Ozcan
Siemens Building Technologies CPS Software Hub
Chicago, USA
murat.ozcan@siemens.com

***Abstract*—The overall purpose of this study is to describe the new paradigms achieved in testing, in lieu of the transition of applications from the desktop, to web and to cloud. We study the Siemens Building Technologies Horizon Cloud Program, where products are enabled by the Horizon ecosystem with a focus on speed, independence and small complexity. Horizon Cloud provides widespread connectivity to legacy & new controls, 3rd party, IoT, edge devices and a step-wise covering of on-site, real-time requirements. Horizon Cloud features its own apps based on building integration framework, along with themes, with an open API for ecosystem and SSP. The program also provides a common data model and API to BT Building Integration Framework to map southbound data against model.**

We start by describing the architecture of the Horizon Cloud program; the components that make up the whole, how they relate, technologies being used in each layer and how they are being tested individually and as a whole.

We continue by describing the continuous delivery process; Dockerization of the continuous integration pipeline, the architectural components being built as Docker images and being run as Docker containers, being deployed and tested in the cloud at unit, API and UI levels. Later, we describe how the components that make the architecture are orchestrated in AWS hosted OpenShift.

We describe the abstraction and the structure of the test code, the page object model, how specifications are separated from the page object / service / logic layer and the test data.

Finally, we analyze three applications of combinatorial testing (CT) in this modern development environment, incorporating the CT model into automation using BDD style Protractor end to end UI tests. We go through the CT modeling process using newly released, open source, cloud based CTWedge: Combinatorial Testing Web-based Editor and Generator. We describe how the model translates into functions in the code and its utilization in behavioral driven tests.

Finally, we analyze a scenario where a sequence of actions are incorporated into a CT model; with a focus on verification of these sequences, compositions of the actions and streamlining the expected assertions, in line, per the data-manipulated test oracle.

***Index Terms*—Combinatorial testing, Input parameter model, Sequence, Angular, Protractor, UI automation**

TODO: Add comments from Rcik Rick : A number of things I think would be of particular interest. One would be your comments on how CT fits into container/Docker development and some of the other platforms that are used a lot today. Most researchers don't know a lot about modern development environments, so your experience and recommendations on how to use CT for these would be very valuable, e.g., what sort of extra tooling would be helpful, characteristics that are good or poor fits for existing CT tools.

Also of interest is the way you integrate covering arrays and sequence testing. This is something that isn't always handled well, and yet very important for an awful lot of applications. The measurements of coverage at the end of the presentation are also of interest, in particular how to use coverage to determine when it's cost effective to complete testing.

I. Introduction

Modern web development typically falls into a design pattern. In the front-end there can be a JavaScript (JS) framework, some popular ones being Angular, React and Vue. The back-end preferences are varied, there is a plethora of choices in language : JS (NodeJS), GoLang, Python, C#, Java to name a few. This is coupled by a database, examples include MySQL, MongoDB, Elasticsearch.

Previous to cloud technologies, such an application would be hosted on a dedicated web server, or a hosting company. In contrast, cloud computing adopts a concept called "virtualization," where hardware resources can be further optimized through software functionality. As a result, not only application performance is optimized but also hosting the application is more cost effective.

A challenge for cloud computing is the resource intensive operating system (OS) usage, where the size of the OS image can be in gigabytes while the application is much smaller. Consequently virtual machines, sinfce they have to host an OS, do not solve this problem. Containerization is one proposed solution, and Docker is one example of a program that performs operating-system-level virtualization. TODO: insert Docker reference In containerization, a layer between OS and applications is introduced to optimize resource usage and eliminate the need for an OS.

This is highly valuable for application development because it enables the application be hosted in a minimal, resource and cost effective "container" which allows the application to be built, deployed and tested faster. Throughout the paper we will refer to this paradigm as Cloud Computing.

In this paper, we will study how Combinatorial Testing (CT) fits in the front-end test automation and continuous deployment of cloud computing paradigm. Examples will include modeling of the input parameter model (IPM), how the model translates to fields and methods in page objects TODO: insert reference, and the utilization of it all in behavioral driven test specifications. One example will include a scenario where a sequence of actions will be incorporated into a CT model, a problem that has been addressed in a variety of ways in previous works.

II. Related Work

A. Introduction to Combinatorial Testing

TODO: @Ludwig: small intro to CT.

B. Previous works on sequences

The method of modeling sequenced parameter groups applied in this paper borrows ideas from a combination of the modeling patterns found in [1]. These modeling patterns include:

1) *Optional and conditionally-excluded values:* the use of N/A value for parameters that are not a part of the parameter group in the current sequence.
2) *Ranges and boundaries:* the reduction of parameter values for certain parameters with over 100 possibilities
3) *Multi-selection & Order and padding:* a variation of these ideas was used for control parameters which enable/disable the parameter groups per the sequence.

C. Previous works on page objects

There has been hundreds of papers on BACnet since early 90s. While many papers have touched on testing. Generally, the testing of Event Enrollment Objects has not been a topic of focus. To get a better understanding on BACnet and Event Enrollment, some of the previous work can be reviewed in [2], [3], [4] and [5].

D. Previous works on cloud computing

There has been hundreds of papers on BACnet since early 90s. While many papers have touched on testing. Generally, the testing of Event Enrollment Objects has not been a topic of focus. To get a better understanding on BACnet and Event Enrollment, some of the previous work can be reviewed in [2], [3], [4] and [5].

III. The System Under Test

A. Description of the Architecture

Horizon Cloud program is composed of many teams and microservice architectural applications. The application / system under test in this study will be Building Operator IC (BOIC). BOIC is used for TODO: describe BOIC , also insert what IC means . BOIC is being development by the Chicago team at Siemens Software Hub.

Figure 1 represents the BOIC architecture. The front-end is an Angular framework, in TypeScript. Note that, this study will focus on black box test automation of the front-end. Testing at different levels of the architecture is planned to be studied in a future paper. The back-end is an ExpressJS application on top of NodeJS platform. At the time of this study, there is an effort to move some of this functionality to other microservices - implemented in GoLang (Go) - to reduce operating costs. The Protocol Adapter (in C#) and Gateway (in NodeJS and Java) serve the purpose of exposing Siemens or third party edge-devices to the cloud. This enables the hardware and the gateway at a customer site to be controlled from a web browser, anywhere in the globe.

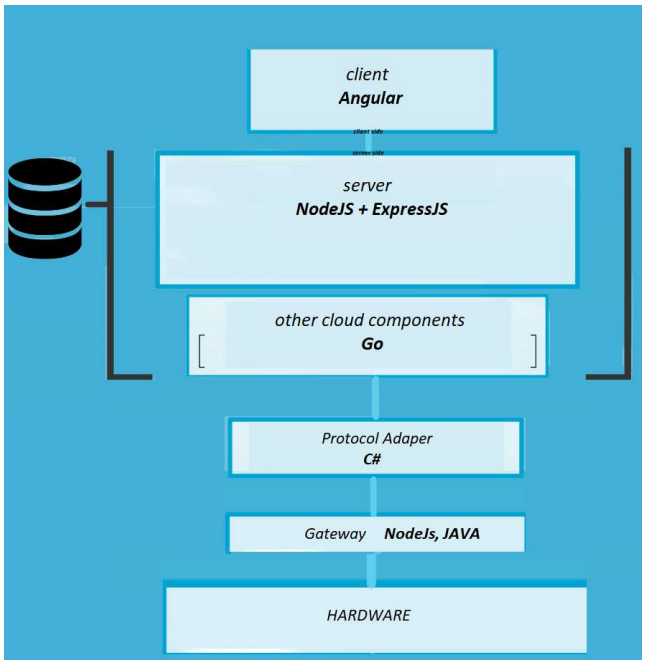

Fig. 1. BOIC architecture

B. Description of the Continuous Deployment Environment

In a traditional configuration management scenario, there may be a plethora of servers needed for web development activities. Staging, committing, packaging the developed code, testing at each stage, source controlling and finally publishing at production server(s) may take place on a number of machines. This can end up in high costs of operation and maintenance in order to enable development activities.

In cloud computing, utilizing containerization, each one of these development activities can be operated in a docker container. Figure 2 shows the GitLab pipeline for the UI component developed in Angular. GitLab is a continuous integration tool that supports the teams by hosting code repositories (similar to GitHub), providing defect tracking, enabling code reviews and continuous integration support. Another example of a continuous integration tool is Jenkins, which can serve the same purpose thorough is extensions.

The team uses Git for source control. TODO: about git. When any code is submitted, the application is built in a docker container, OSS Clearing for licensing is executed, unit testing and linting are done, the application is deployed to an cloud-hosted container and automated UI tests are run targeting the deployment. If testing passes, the code can get merged to the master branch. This automated testing process ensures that after any code commit, the application is tested end to end and the quality is ensured. To give a sense of scale, at the time of writing, the front end is over 15k lines of TypeScript code in Angular, over 1k lines of unit tests in Karma, over 4k lines of end to end (e2e) UI tests Protractor. There are over

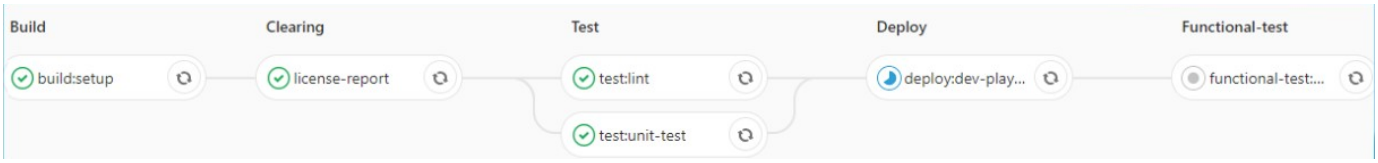

Fig. 2. UI component pipeline

200 unit tests that execute in seconds, over 300 e2e UI tests that execute under 10 minutes.

C. Description of the Test Code Structure

1) Page Object Pattern: Released in 2011, Angular is popular front-end JavaScript framework for designing dynamic web applications. It is maintained by Google with the help of the open source community. It recommends the use of Microsoft's TypeScript language, which introduces class-based object oriented programming, static typing and generics. TODO: refer to wiki.

Components are the building blocks of Angular applications and they easily nest inside each other. Each component may have its own class, html and css file. This structure provides a way to design dynamic web applications while keeping the front-end code clean.

Page object pattern is a popular UI test automation design pattern in the industry. TODO: refer to Fowler and selenium paper. In Fowler's proposed terminology, we refer to web pages as 'pages' and Angular components as 'panels'. Each panel is a class. A page can be made up of many panels, which have a class-based object oriented structure.

Designing the test automation architecture, it became clear that it would not only be straightforward but also efficient to replicate the structure of Angular components in page objects as pictured in Figure 3. One advantage of this is easier maintenance of the automation code as the development code changes. The other advantage is its benefits with asynchronous test execution.

In a panel object, the element selectors become the fields and the methods become test actions within the component / panel. TypeScript is a superset of JavaScript. It complies into JavaScript and includes the latest EcmaScript features TODO: reference to EcmaScript . The latest EcmaScript features allow developers to write sequential looking asynchronous code.

With regards to test automation this means clean code that executes as fast as the environment allows, resistant to flakey tests and stale elements on the page. This is because the page components load with panel-object-classes simultaneously, while the page element selectors get instantiated. We plan to study this design pattern in a future paper. For the purpose of the current study, it is sufficient to understand that all test relevant methods are housed in classes that represent panel objects / Angular components.

2) Test data: The test data is stored in JSON files. This allows parameterization of test inputs, input-driven tests, as well as ease of maintenance as the UI changes. Another perk of using JSON is being able to convert test suites generated with CT tools - ex: ACTS, CTWedge TODO: references - from csv format to JSON format.

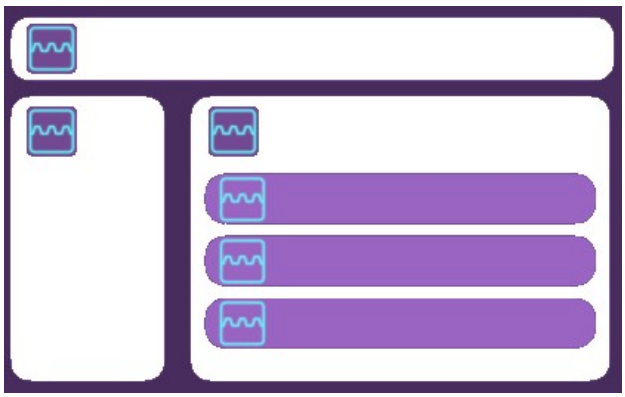

Fig. 3. Angular components as page and panel objects

3) Test Specifications: The team uses Protractor test framework, which is the default end-to-end test framework for Angular applications. TODO: insert reference It runs tests against application running in a real browser, interacting with it as a user would. It combines technologies from Selenium webDriver, NodeJS and allows tests to be described in a BDD format -*Given, When, Then*-, describing the overall behavior of a system at a low level.

The tests specifications are stored in Protractor spec files written in TypeScript. The aforementioned benefits of the technology stack allow developers in test to write clean, simple, synchronous looking asynchronous code, without hard-waits or sleeps, resistant to stale elements and flakey tests.

Over time during our development, the test specification came to be described in State-based, Behavior-driven Acceptance Scenarios TODO: reference blog post. Generally, these are in the format:

Given I have arrived in some state
When I trigger a particular event
Then the application conducts an action
And the application moves to another state

It was found out that this style of expression is not only descriptive at a meta level, but also maps well to the page object pattern. Figure 4 and Figure 5 show an a Protractor test spec sample with a state-based, behavior-driven acceptance scenario. As observed from the sample code, *tunnelPage* class / page-object is utilized for:

- setup, using the *setDropDown()* class method.
- test action, using the *startButton* class field.

```
beforeAll(async () =>{
  await tunnelPage.setDropdown();
});
describe(`Workflow: progress with tunnel connection until it is active\n
  Given: I have arrived at state Creating Tunnel Connection with start button enabled`, () => {
  it(`When I trigger an event:  Start button is clicked,
    Then: I expect the application to move to Creating Tunnel Connection state\n`, async () => {
    await tunnelPage.startButton.click();
    expect(tunnelPage.tunnelConnectionStateText.getText()).toBe('Creating your Tunnel connection...');
  });
});
```

Fig. 4. Protractor test code sample in State-based, Behavior-driven Acceptance format

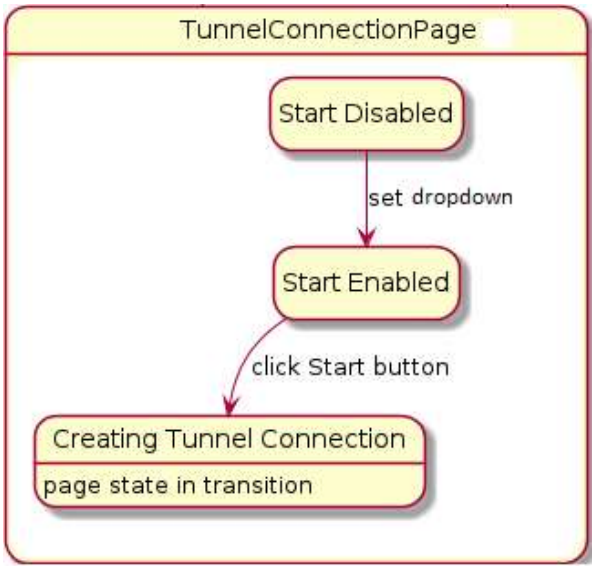

Fig. 5. State-based, Behavior-driven Acceptance Scenario

- test assertion, verifying that *tunnelConnectionStateText* class field / selector shows the correct text.

The code also samples the Protractor BDD style where a test workflow scenario is covered. The *it* block is one test in the workflow, the *describe* block houses more tests / *it* blocks that lead to the completion of the workflow - not shown in this example for the sake of simplicity.

IV. Test Methodology

In the following sections the coverage measurements, the pro's and con's of the IBM and the SBA methods will be evaluated.

A. Input driven testing of hardware

Considering t-way coverage measurement, SBA method provides evidence of the measurement while coverage measurement in IBM method is inconsequential as explained in the previous section.

Regarding the Application of the methods, IBM method is straightforward once the constraints are figured out. SBA method has elaborate steps as outlined in section III D. In step 3 of the method (plug-in construction), there is a need for a script that replaces integers with real values. Currently a freely available tool does not exist for such a task.

Constraints are important for both methods, but they are vital for IBM method. They can be manageable in a small scale, yet as the SUT gets larger they get more complicated. Figure **??** shows the Javascipt code for constraints used in the IBM method.

Considering the above factors, IBM method may be preferred for smaller systems under test while SBA method may be preferred for larger systems. However, the significance of the above factors for the system and/or the test team may effect the preferences for the method of use.

B. Filtering Points by Tag Combinations

C. State Transitions Between Combinations of Alarm Data, With a Variable Oracle

D. Sequenced Filtering, Searching, Sorting of Geo-located Sites

V. Conclusion and Future Work

The initial challenge in this study was how to best test EEO configurations in the most efficient manner. During the study it became apparent that the methods applied can be used to test any system where the parameter groups for the Combinatorial Input Parameter Model (IPM) are not simultaneously available, and instead may appear sequentially. Future work on this front may incorporate system test ideas where each seed (in the SBA method) or different color (in the IBM method) being a different module of functionality.

Acknowledgments. This research was carried out in the context of Austrian COMET K1 program and publicly funded by the Austrian Research Promotion Agency (FFG) and the Vienna Business Agency (WAW).

REFERENCES

[1] I. Segall, R. Tzoref-Brill, and A. Zlotnick, "Common patterns in combinatorial models," in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 624–629.

[2] S. T. Bushby, "Bacnettm: a standard communication infrastructure for intelligent buildings," *Automation in Construction*, vol. 6, no. 5-6, pp. 529–540, 1997.

[3] ——, "Bacnet today: Significant new features and future enhancements," *ASHRAE journal*, vol. 44, no. 10, p. S10, 2002.

[4] R. Nussbaumer and G. Waters, "Bacnet support for epics."

[5] H. M. Newman, "Bacnet celebrates 20 years," *ASHRAE Journal*, vol. 57, no. 6, p. 36, 2015.