Crazy title

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

***Abstract*—The overall purpose of the presentation 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. We also describe the test design, test modeling process, UML in relation to test functions and its close proximity to the test code in the repository.

Finally, we analyze three applications of combinatorial testing (CT), incorporating the CT model into automation using BDD style Jasmine framework's 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.

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.

***Index Terms*—Combinatorial testing, Input parameter model, Sequence, EEO, BACnet**

I. Introduction

Building Automation and Control Networks Protocol (BACnet) is an interoperable communication protocol for building automation and control networks. When a piece of hardware or software is BACnet compliant, this means it can communicate with the hardware and software of any other BACnet compliant vendor. BACnet compliance is an essential aspect every research and development project at Siemens Building Technologies as well as any competitor such as Honeywell, Johnson Controls, Schneider Electric etc.

II. Related Work

A. Previous works on BACnet

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 [1], [2], [3] and [4].

B. Introduction to Combinatorial Testing

TODO: @Ludwig: small intro to CT.

C. Description of IBM-Method

The method of modeling sequenced parameter groups applied in this paper borrows ideas from a combination of the modeling patterns found in [5]. 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.

D. Description of SBA-Method

In [6] a combinatorial methodology to model composed software systems is presented. The main idea behind this methodology is to combine legacy test suites of components of an SUT, via a so called *meta-array*, to get a test suite for the whole SUT. It is also shown that, if the legacy test suites, as well as the meta-array fulfill certain combinatorial criteria, so does the test suite for the whole SUT (for details see [6].) The generation of a combinatorial test suite for an SUT with the TODO: to be defined in into to CT: IPM $(v_1, \ldots, v_g)$ can be described as follows:

Step 1: Partition the given parameter configuration $(v_1, \ldots, v_g)$ into classes $V_1, \ldots, V_k$.
Step 2: For each $i = 1, \ldots, k$ construct legacy test suites that are covering arrays of a certain strength t: $S_i = MCA(N_i; t, |V_i|, V_i)$.
Step 3: Construct a meta array $\mathcal{M} = MCA(N_m; t, k, (N_1, \ldots, N_k))$.
Step 4: For all columns of $\mathcal{M}$ do the following: in column j of $\mathcal{M}$ replace all occurrences of n with the n-th row of $\mathcal{S}_j$.

The result of this construction is again a covering array of strength t.

III. Modelling of BACnet

A. Description of BACnet

A BACnet Event can be characterized as any change in the value of any property of any object that meets a particular criteria. The purpose of an Event Enrollment Object (EEO) is to define an event and offer the engineer an association with the occurring event and the transmission of notification messages.

Vendors and building control products of those vendors may have varying implementations for EEO configurations in the user interface (UI). Until recently, the Siemens legacy building control product, Siemens APOGEE Insight Workstation, could have above 5 million ways to configure an EEO. Due to its complicated nature the EEO configuration was previously left open for the user and most of these configurations could be invalid. If the configuration was invalid at the workstation, once the EEO was downloaded the field panel, it would show a configuration error for the EEO. The EEO would then need to be modified at the field panel or it would have to be modified at the workstation and re-downloaded to the panel.

Recent enhancements made the correct configuration of EEOs enforced and streamlined in the UI of the workstation, where one EEO at a time can be configured and then later downloaded to the field panel. Another enhancement, the Harmonization Tool, was introduced to provide the ability to mass-configure many EEOs in Excel and feed them to the field panels.

These enhancements created the need to test correct configuration of EEOs at the workstation and the mass configuration alternative to the workstation; the Excel based Harmonization Tool. The answer was to create a meta test suite that would be utilized for round-trip testing EEOs in the system. For practical testing of BACnet, one needs to be able to mass-feed EEOs into the system somewhere (Harmonization Tool), be able to move it around in the system, finally verify configuration authenticity at the panels and the workstation. Figure **??** displays the work flow of testing EEOs.

1) Test suites are fed to the Harmonization Tool.
2) The EEO configurations are pushed/downloaded to the field panels, where the field panels should show configuration OK if the EEOs were configured correctly.
3) Finally the EEO configurations can be bulk-uploaded to the workstation from the panel. EEOs should show configuration OK if they were configured correctly.

B. Modeling BACnet EEO's with IBM-Method

In software not all parameters are available simultaneously: some parameters can be enabled over time based on user choices, and these parameters can have covering arrays of their own values [7].

In Figure **??** you see a color coded, pseudo input parameter model (IPM) of BACnet EEO configuration in *Siemens APOGEE Insight Workstation*. Here, the green path signifies a group of related parameters: ObjectType, ObjectPropertyReferenceAIAOAVBOMO, SetPointOrFeedbackPropType, SetPointOrFeedbackPropProperty. Here ObjectPropertyReferenceAIAOAVBOMO with true/false value is used to control parameter group sequences as mentioned in section II B item 3. The parameters relevant to the sequence are allowed to have values, while the parameters in other sequences are disabled by assigning a N/A value as mentioned in section II B item 1. The parameter SetPointOrFeedbackPropProperty which had 208 possible values was reduced utilizing ideas from section II B item 2. Value partitions that had similar uses, for example ASHRAEproperty223 to ASHRAEproperty334, were treated as a single partition and the boundaries were utilized. Value partitions that were variations of a common logic were also grouped together and single values were chosen to represent those partitions. These represented cases that were equivalent to each other with respect to the testing scenario [5]. At a final step of refinement, it was ensured to cover the values users most commonly utilize. These represented the corner cases that need to be captured by the model and tested explicitly [5]. The effort reduced the possible values of this parameter from 207 to 37.

Refer to figure **??**. Related parameter groups are henceforth referred to by color groups. In the figure the green path is described as when the Object Type is either analog input (AI), analog output (AO), analog value (AV), binary output (BO) or multisate output (MO). When this is the case the related fields are enabled and the others are disabled. The teal path signifies another group of related parameters: binary input (BI), binary value (BV). The blue path with multistate value (MV) works with the same logic.

The 'enabling' of related parameters occurs as the parameter used for control of that parameter group is set to TRUE. Since only a single parameter group should be enabled at a time,

through constraints, the other control parameters which control other groups are set to FALSE. In turn, the parameter groups that are controlled by those control parameters are forced to have values equal to N/A. Here the use of N/A value for parameters that are not a part of the active parameter group in the current sequence is an idea inspired from the modeling pattern referred to in section II C 1. The use of control parameters with values TRUE & FALSE, the use of the control logic to identify the sequence in which the parameter group is occurring are ideas inspired from the modeling patterns referred to in section II C 3.

A better way to represent this sequence logic would be to separate them in blocks. Figure **??** pictures how this can be structured. These blocks represent different UIs the user is presented with based on their parameter choices for the base block. The base block contains the parameters ObjectType and EventType, which are common to all paths. ObjectType's parameter values AI, AO, AV, BO, MO enable block 1 highlighted in green. ObjectType's parameter values BI and BV enable to block 2 highlighted in teal. ObjectType's last parameter value MV relates to block 3 highlighted in blue.

Block 1 (in green) contains the parameters SetPointOrFeedbackPropType, SetPointOrFeedbackPropProperty.

Block 2 (in teal) contains the parameters ObjectPropertyReferenceMV, PropertyStateType, ValueUnsigned.

Block 3 (in blue) contains the parameters ObjectPropertyReferenceBVBI, PropertyStateType, ValueBinary.

Notice PropertyStateType being common in both blocks. Depending on the choices in ObjectType of the base block, a default parameter value gets selected for PropertyStateType: binary when ObjectType is binary-input or binary-value, Unsigned when ObjectType is Unsigned. We will exemplify the 3 different sequences that may occur based on the initial choices of the user in the base block.

Example 1: User selects ObjectType = 'AO', EventType = 'floating-limit' or 'out-of-range', the next set of configurations can only be for SetPointOrFeedbackPropType (over 2 choices under it) and SetPointOrFeedbackPropProperty (25 choices under it). All other parameters set to N/A utilizing constraints.

Example 2: User selects ObjectType = 'MV', EventType = 'change-of-value', the next set of configurations can only be for ObjectPropertyReferenceMV (True or False), PropertyStateType (only Unsigned), ValueUnsigned (4 configurations). All other parameters set to N/A utilizing constraints.

Example 3: User selects ObjectType = 'BV', EventType = 'change-of-state', the next set of configuration con only be for ObjectPropertyReferenceBVBI (True or False), PropertyStateType (only Binary), ValueBinary (00, 01, 10, 11). All other parameters set to N/A utilizing constraints.

A better understanding can be achieved by observing the verbal constraints. The Javascript code will be revealed in a later section while analyzing the pros and cons of the method.

If ObjectType = AI,AO,AV then EventType = floating-limit OR out-of-range

If ObjectType = BO,MO then EventType = command-failure

If ObjectType = MV then EventType = change-of-value

If ObjectType = AI,AO,AV,BO,MO then ObjectPropertyReferenceAIAOAVBOMO = TRUE ObjectPropertyReferenceMV = FALSE, ObjectPropertyReferenceBVBI = FALSE, PropertyStateType = N/A, ValueBinary = N/A, ValueUnsigned = N/A

If ObjectType = BI,BV then EventType =change-of-state, ObjectPropertyReferenceBVBI = TRUE, PropertyStateType = Binary, ObjectPropertyReferenceAIAOAVBOMO = FALSE, SetpointOrFeedbackPropType = N/A, SetpointOrFeedbackPropProperty = N/A, ObjectPropertyReferenceMV = FALSE, ValueBinary != N/A, ValueUnsigned = N/A

If ObjectType = MV then EventType = change-of-value, ObjectPropertyReferenceMV = TRUE, PropertyStateType = Unsigned, ValueUnsigned != N/A, ObjectPropertyReferenceAIAOAVBOMO = FALSE, SetpointOrFeedbackPropType = N/A, SetpointOrFeedbackPropProperty = N/A, ObjectPropertyReferenceBVBI = FALSE, ValueBinary = N/A

C. Ideas to reduce parameter values

In addition to equivalence-partitioning parameter values in block 1 (refer to block 1 example from the previous section)

While configuring for a MV (refer to block 2 example from the previous section) another idea came to mind. For the parameter ValueUnsigned, instead of many possible configurations, it appeared that a covering array can be utilized. Figure shows the UI for what configures the parameter named ValueUnsigned in the IPM: when the object of concern is an MV, the PropertyStateType is set to Unsigned and these are the values the user is presented with for the configuration.

D. Modeling BACnet with SBA-Method

In [6] a combinatorial methodology to model composed software systems is presented. The main idea behind this methodology is to combine legacy test suites of the components, via a so called *meta-array*, to get a test suite for the whole SUT. It is also shown that, if the legacy test suites, as well as the meta-array fulfill certain combinatorial creteria, so does the test suite for the whole SUT (for details see [6].)

The SBA-Method [6], can be summarized in 5 core steps. Each step will be detailed in subsequently.

1) Generate seed covering array (CA) for strength t=1 for each of the parameter partitions: We start by dividing the interrelated parameter groups into seeds. TODO: Ludwig, please insert a short description of what seeds are, perhaps make a reference to a paper, maybe this one? [6] In identifying the seeds for an EEO, we borrow initial ideas from the IBM method for separating the sequenced parameter groups. The parameters and values that make possible the IBM method; parameters that enable/disable blocks (using TRUE and FALSE), parameter values that are 'N/A' which disable blocks, are not needed. Refer to the IPM in Figure **??**. The second important point is that each seed gets its unique parameters whether they are common in the other seeds or not: in the figure these are ObjectType and EventType parameters which are common in the seeds but their parameter values are different based on user selection. For example in seed2: if ObjectType is BI or BO, EventType is set to changeOfState

immediately and the ValueBinary can have values from 00 to 11. Another example is in seed3: if ObjectType is MV, EventType is set to changeOfValue and ValueUnsigned is one of the 8 configurations, of which we are using 4 as described in Figure **??**. Once the seeds are defined, we start generating a CA for strength t=1. In Figure **??** 1-way covering arrays are shown in ACTS screenshots: seed1 has 37 tests, seed2 and seed 3 have 4 tests each.

2) Generate a meta CA of strength t=2; the input is an IPM where each parameter corresponds to a partition and the parameter values are integers from 1 to the size of the respective seed CA (from step 1) : Here seed1 corresponds to partition M1, seed 2 to M2 and so on. Each test in the seed is represented by an integer: 37 in M1, 4 in M2 and 4 in M3. Following this a meta CA is generated of strength 2 (t=2 : 2-way). The meta array in our example has 148 tests as shown in the ACTS screen shot in Figure **??**

3) Expand step 1 and 2 with real values instead of numbers: apply the plug-in construction: : In this step we take our meta array with 148 tests and expand it with real values. In Figure **??** shows the entirety of parameters: M1: ObjectType, EventType, SetPointOrFeedbackPropType, SetPointOrFeedbackPropProperty; M2: ObjectType, EventType, ValueBinary; M3:ObjectType, EventType, ValueUnsigned . Parameter M1 has 37 possible values used in 148 tests (refer to step 1 and 2), so the integers 1 to 317 get replaced with the tests in step1. Parameter M2 has 4 possible values in 148 tests, so the integers 1 to 4 get replaced with the tests in step1 and so on for M3.

4) Generate juxtaposed CAs of strength 2 for each of the partitions: : Refer back to step1 where we created seed CAs for strength t=1 (seed out) as shown in Figure **??**. In this step we have a parallel effort where we create the same CAs with strength t=2 (jux out: juxtaposed out) instead. In Figure **??** these 2-way CAs can be observed: 925 2-way tests for seed1, 8 2-way tests for seed2 and 4 2-way tests for seed3.

5) Vertically juxtapose the juxtaposed CAs in step 4 to the result of step 3. This guarantees that the final result is again a 2-way covering array: : We simply add the 2-way CA under the 1-way CA as shown in Figure **??**. Here the first seed has 925 tests while the others have 8 and 4 respectively, this may seem challenging at first. For seed 2 and 3 we simply, repeatedly keep adding in blocks (of 8 and 4 respectively) until we cannot add anymore. If the final addition is not a multiple of 925, then we add the remainders.

IV. Experiments

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

A. Coverage measurement comparison

All t-way coverage measurements were made using NIST's coverage measurement tool Combinatorial Coverage Measurement (CCM) command line version [8].

For IBM method, the coverage with constraints factored in and the coverage without constraints are examined in figures **??** and **??**. Figure **??** shows the 2, 3, 4-way coverage measurement of the IBM method, respectively figure **??** shows the coverage measurement without constraints.

As mentioned in Section III B, the use of constraints are at the foundation of the IBM method. The 2-way coverage without constraints is at 100% as seen in figure **??**. The application of constraints, which make the method possible, reduces the 2-way coverage measurement to 58.1%. Because constraints are an integral part of the method, in this evaluation we believe that the coverage measurement of the IBM method is inconsequential.

For SBA method, the coverage with constraints factored in and the coverage without constraints are examined in figures **??** and **??**. Figure **??** shows the 2, 3, 4-way coverage measurement of the SBA method, respectively figure **??** shows the coverage measurement without constraints. The 2-way coverage without constraints is at 100% as seen in figure **??**. The minimum 2-way coverage without constraints is also at 100%.

The significance and the impact of minimum t-way coverage on branch coverage conditions in the code is discussed in paper [9]. Label M as the minimum 2-way coverage; i.e., the lowest proportion of settings covered for all t-way combinations of variables. For example, 2-way combinations of binary variables have four possible settings: 00, 01, 10, 11. Some variable pairs may have all four settings covered, but others may have less. M is the smallest proportion of coverage among all of the t-way variable combinations. M is also viewable as the rightmost line in the coverage strength meters in Figure 1 and 3.

Applying the constraints to the SBA method has a miniscule impact on the overall 2-way coverage: down from 100% to 99.76% as seen in figure **??**. The effect on 3 and 4-way coverage is similar. TODO: Ludwig, do you want to insert a sentence on why this is? The impact of constraints on the minimum 3-way and 4-way coverage is non-existent, however on minimum 2-way coverage the difference is 40%. TODO: Ludwig, do you want to insert a sentence on why this is?. Overall, t-way coverage analysis is valid on the SBA method.

B. Pro's and Con's

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.

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 the Austrian COMET K1 program and publicly funded by the Austrian Research Promotion Agency (FFG) and the Vienna Business Agency (WAW).

References

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

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

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

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

[5] 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.

[6] L. Kampel, B. Garn, and D. E. Simos, "Combinatorial methods for modelling composed software systems," in *Software Testing, Verification and Validation Workshops (ICSTW), 2017 IEEE International Conference on*. IEEE, 2017, pp. 229–238.

[7] R. Kuhn, Y. Lei, and R. Kacker, "Practical combinatorial testing: Beyond pairwise," *It Professional*, vol. 10, no. 3, 2008.

[8] D. R. Kuhn, I. D. Mendoza, R. N. Kacker, and Y. Lei, "Combinatorial coverage measurement concepts and applications," in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 352–361.

[9] D. R. Kuhn, R. N. Kacker, and Y. Lei, "Measuring and specifying combinatorial coverage of test input configurations," *Innovations in systems and software engineering*, vol. 12, no. 4, pp. 249–261, 2016.