Automated Metadata Construction To Support Portable Building Applications

Arka A. Bhattacharya
UC Berkeley
arka@eecs.berkeley.edu

Dezhi Hong
University of Virginia
dh5gm@virginia.edu

David Culler
UC Berkeley
culler@cs.berkeley.edu

Jorge Ortiz
IBM Research
jjortiz@us.ibm.com

Kamin Whitehouse
University of Virginia
whitehouse@virginia.edu

Eugene Wu
Columbia University
ewu@cs.columbia.edu

ABSTRACT

Commercial buildings consume nearly 19% of delivered energy in the U.S, nearly half (42%) of which is consumed in buildings with digital control systems [23] comprised of wired sensor networks. These sensors have scant metadata, and are represented by "tags" which are obscure, building-specific and not machine parseable. We develop a human-in-the-loop synthesis technique which uses syntactic and data-driven steps to parse these sensor tags into a common namespace, which can enable portable building applications. We show that our technique allows an expert to fully parse a large fraction (70%) of the tags with 24, 15 and 43 examples for three large commercial buildings comprising 1586, 2522 and 1865 sensors respectively, and deploy three portable applications on two buildings with less than 30 examples.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*Program Synthesis*; H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*

Keywords

Building Management Systems; Sensor Metadata; Portable Building Applications; Metadata Normalization

1. INTRODUCTION

While advances in cyberphysical systems have provided new infrastructures for monitoring and interacting with physical environments, traditional automation and control infrastructures dominate the building stock and must also advance. The monitoring and actuation networks that are wired into commercial buildings, industrial plants, and urban infrastructures for their basic operation are increasingly accessible through the BMS (Building Management System) or SCADA (Supervisory Control and Data Acquisition) systems that host higher level control, retain historical data, and provide visualization. Many of these systems provide some kind of programmatic interface to the sensors, actuators, and historical data under their management [2, 7, 24]. But, whether provided by novel networks or legacy instrumentation, extracting meaningful information from sensor data and taking actions based on that data depends fundamentally on the metadata available to interpret it. While development of effective metadata schema has become an active topic for emerging systems, it has long been a core challenge in the legacy setting. Often, a critical step in the deployment and engineering of large automation or building systems is formulating consistent naming conventions so that the many aspects of a "point" — its function, type, position, role, and so on — are represented in its "tag", [1] typically a highly constrained alphanumeric string (cf, ([9, 4]). These encodings are often quite sophisticated , as they have to convey many distinct attributes and relationships, i.e., metadata, in a compact representation that is interpreted by various engineers over many years.

However, this terse metadata is designed to be used by specially trained engineers in the field; it is not designed for machine translation. Typically, tags are attached to various screens as part of the human-machine interface of BMS and SCADA systems, so engineers can check status and plot trends. With knowledge of the intention of the naming scheme and the ad hoc association to various views, the syntax and semantics of the tag are aparent to the well-trained engineer or facilities manager. But, developing an algorithm to parse the tag and soundly identify each of the semantic attributes in it is an altogether different story. There may be no field delimiters or multiple, spurious ones; symbol definition may be context dependent; different schemas may encode the same type of sensor; and each vendor or each deployment may follow different rules.

Thus, even with programmatic access to tags, data, and other descriptive information, scaling analytics or intelligent control across the commercial building stock to, say, improve energy efficiency is likely to be intractable, as long as the

BuildSys'15, November 4–5, 2015, Seoul, South Korea..
© 2015 ACM. ISBN 978-1-4503-3981-0/15/11 ...$15.00.
DOI: http://dx.doi.org/10.1145/2821650.2821667.

[1] An example of such a tag would be `BLA1R465__ART` which denotes the sensor point is in *site* `BLD`, is part of the air handling unit (*ahuRef*) `1`, located in room(*zoneRef*) `465`, and is a *zone air temp sensor* (denoted by `ART`). We term the labels *site, ahuRef, zoneRef, zone air temp sensor* as **fields**, and the corresponding substrings (`BLD`, `1`, `465`, `ART`) as their **field-values**.

basic steps in interpreting the metadata involve labor intensive manual efforts by highly trained professionals with deep knowledge of each building. Sophisticated applications may be developed for a particular building, but require customized building-specific logic and queries, which are not portable or scalable across buildings.

There have been extensive efforts to standardize and automate of the management of sensor metadata in SCADA and related systems. However, tag naming remains heterogenous and inconsistent between commercial vendors/agents [25]. Some tools (e.g [5, 10]) have been created to automate the generation of tag namings, but are largely oriented to making the tags more "human readable" for later manipulation rather than making them interpretable by computers for direct analysis.

In this paper we develop an automated synthesis technique that learns how to transform and *normalize* legacy tags into a well-formed representation (cf., [1]) using a small number of examples from an expert, e.g., the building manager. Such building managers understand the tags, but they are unlikely to be adept at writing complex regular expression programs to transform them to a common, understandable namespace. The transformation to such a namespace yields *semantic relationships between sensors*, which enables analytics applications to be deployed without a priori building-specific knowledge.

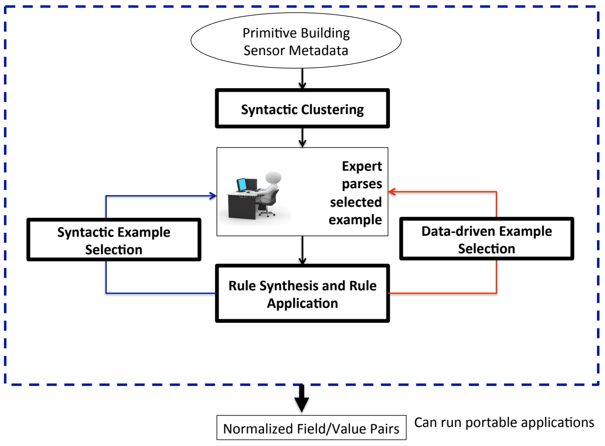

Primitive Building Sensor Metadata
Syntactic Clustering
Expert parses selected example
Syntactic Example Selection
Data-driven Example Selection
Rule Synthesis and Rule Application
Normalized Field/Value Pairs
Can run portable applications

Figure 1: High-level view of major steps in our Algorithm

We draw inspriation from programing-by-example techniques developed in [13]. To our knowledge, this paper is the first attempt to apply these techniques to buildings, and find that the techniques must be fundamentally rethought to work well in building systems. Our approach is shown in Figure 1. To reduce the complexity of parsing the varied and inconsistent schemas contained within a building, we first cluster the sensor metadata into chunks which are more likely to share a common schema. An example is selected from one of the clusters and provided to an expert to parse into a common namespace (In practice, a simple GUI is provided for this). Based on the example parse, we synthesize rules from a domain-specific language which are consistent with the expert-provided example. We then apply these rules to parse the field encodings of the remaining sensor tags in that cluster, i.e., to generalize the example to a program that can parse many of the tags. Based on the resultant parsing, a new example is selected to be presented to the expert, and so on. Ideally, a few iterations of this example-driven synthesis loop should produce rules that correctly parse a large fraction of the tags. In practice, this is the case, but several important factors and subtleties arise, which we study in the sequel.

Once progress has been made in qualifying tags through these syntactic methods, if we have access to the data streams for the points associated with the tags, we can employ learning on the data to attempt to find points that are semantically related, but with syntactically distinct tags. This 'boosting' is represented by the second loop in the figure. In either loop, an important question is how to select the next example to present to the expert, since the human in the loop is the precious resource. And, a critical question is when to stop. Typically, most of the tags in a building conform to a few simple encoding formats, but many indiosyncratic formats are present with few tags each. With proper clustering, selection, rule synthesis and generalization, the vast majority of tags are resolved with a few examples, but a long tail of obscure ones remains. On the other hand, the real goal of this process is to enable portable applications on buildings and any such application only requires certain types of points. Thus, we also study how many examples are needed to resolve all the points that are relevant to certain important applications.

The three large commercial buildings used in our study have 1586, 2522 and 1865 sensors respectively and come from completely different institutions with different building systems, installers, and BMS vendors. We find that, indeed, a few examples are sufficient to produce rules to parse a large fraction of the tags in each. Our technique is able to normalize the metadata of 70% of all sensors in just 24, 15 and 43 examples for the three buildings. The synthesis technique is robust enough to handle the presence of obscure and noisily encoded sensor metadata. However, the pre-clustering step is essential in some buildings to avoid over-generalization of the synthesized rules. We study the criteria used to select the next example to present to the expert and find that random selection generally performs better than application-specific heuristics, and choosing a random example from the cluster with the largest number of yet-unqualified tags is robust.

The long tail of obscure tag formats slows convergence; to parse the entire set of tags requires in 161, 116 and 196 examples respectively. However, the applications of interest generally do not require normalizing the metadata of all sensors in a building, but only specific sensor types. That is not to say they are uniformly encoded throughout — in one of our buildings, the zone temperature sensors were encoded six different ways.

The Data-Driven Example Selection builds a random forest classifier for the set of sensors required for an application from the set of sensors already normalized, and applies it to the remaining sensors in the building to identify similar sensors that have not yet been presented to the expert. This classifier is built using a feature vector computed from the physical data associated with the sensors. For three applications on the two of our buildings with accessible data streams the required sensors are parsed with an order of magnitude fewer examples than with only syntactic example selections.

The techniques developed here are likely to be applicable to the other large legacy sensor networks, such as industrial processing, or urban monitoring, and provide a metadata framework that can be adopted without need for such learning-based transformations in emerging sensor networks.

Section 2 provides a more concrete background of the metadata problem, and describes the challenges. We then describe our program synthesis and example selection techniques in Section 3. We evaluate our synthesis technique on three large commerical buildings, and run three portable efficiency applications on them in Section 4. We conclude in Section 5.

2. MOTIVATION AND BACKGROUND

In this section we show the nature of the sensor metadata problem in buildings, the shortcomings of existing approaches, and the challenges involved in synthesizing regular expression programs from expert-provided examples.

2.1 Background

Nearly half (42%) of all commercial buildings in the U.S are equipped with digital control systems [23]. Typically the vendor (e.g JCI, Siemens) contracted to set up the digital control systems of a particular building uses company and deployment-specific guidelines to "tag" sensor points. Often, the only metadata accompanying a sensor stream is its tag ([9]); wherein the vendor and the facilities manager try to encode all the pertinent information for a particular sensor. In our testbed a sensor tag `BLDA1R465__ART` encodes the following information: `BLD` denotes the *site* name, `A1` indicates it is part of the first air handling unit, `R465` indicates it is located in room 465, and `ART` indicates it is an air temperature sensor. Another sensor labelled `BLDA1R465__ARS` indicates that it is in the same room 465 (`R465`), part of the first air handling unit (`A1`) but is a room temperature setpoint (`ARS`). Note, the two example tags not only encode the location and function of the sensors, but also the semantic relationship between them and other sub-systems in the building. These encodings typically vary between buildings (often, even those deployed by same the vendor) — for instance, in another building in our data set, a tag looks like this: `BLD.S2-06:CTL STPT:PRIORITY` with `06` indicating that the sensor was part of the 6th variable air volume unit on the second floor (`S2`), and `CTL STPT:PRIORITY` means that it is an air temperature setpoint.

Such custom, condensed encodings are widespread. We surveyed several different BMS vendors and found many variants of such encodings and no other available metadata for the sensors. This makes it hard to infer a sensor's context uniformly across buildings and precludes the development of applications that can scale across buildings.

2.2 Related Work

There have been various data-driven efforts to capture the contextual and semantic relationships between sensors in order to build applications, such as type classification of sensors [19] and finding spatial relationships between sensors ([11, 15, 17]). However, these techniques either classify sensors into broad categories (type classification), and do not capture semantic (or functional) relations between sensors (e.g which air temperature sensor is related to which setpoint sensor), and hence are not useful in writing applications which depend on the semantic relationships between sensors. Even if techniques like [11, 15, 17] can predict that an air flow sensor and a temperature sensor are in the same room, it cannot determine the air handling system the room is a part of — information often encoded in the sensor metadata tag directly.

There has also been prior work in substring extraction[13], and log record manipulation[16] using examples from a human. However, spreadsheet and log data comprise records encoded in very few schemas or formats, requiring only two or three human examples to synthesize programs to extract all the required fields. These techniques fail to parse building sensor metadata, which present a far more heterogenous and noisy dataset, containing many different schemas and hundreds of encoded fields (comparison shown in Figure 3). We achieve the required robustness using a combination of clustering and domain-specific language contructs. We draw from the boolean classification techniques in [13] to avoid over-generalization of synthesized rules. In the building domain, industrial softwares like PI from OSIsoft [20] give users the ability to generate wildcard regular expressions to select a set of tags. Our domain-specific language provides for much more powerful regular expressions than such software. [22, 21] use string matching to find the most likely fields in building tags. This approach breaks down when fields are represented by only one or two characters, as shown in our previous examples.

Currently, there is no consensus schema in the sensor network, or BMS vendor community about a particular schema to represent and encode all sensors in a building. Some schemas such as Green Building XML [12], and Industry Foundation Classes [6] require a very high level of detail for every metadata tag, making it unsuitable for use in our context where that level of detail might not be available. There have been recent efforts such as sMAP [7], HomeOS [8] and Building Depot [24] to systematically describe sensors and their functionality in a building. However, these systems ignore the problem of mapping the thousands of existing underlying sensors to their desired sensor description templates. We chose the markers/idioms specified in the Project Haystack ([1]) convention as our target namespace.

2.3 Challenges

Our technique automatically synthesizes regular expression programs that transform primitive SCADA metadata into a common desired namespace. Learning from expert-provided examples has two advantages over manually generating regular expression programs — (a) the experts, often facility managers or maintenance professionals, are not well-equipped to construct the correct regular expression programs themselves; (b) inconsistencies in the metadata structure including obscure and noisy tag encodings require hundreds of very complex regular expressions, which would make manual regex generation error-prone, if not impossible.

Unlike machine opcodes, the language of the primitive metadata was not created with the intention of being machine decode-able. Machine opcodes generally have specific fields which specify how to parse a particular sequence of characters/bits (*fixed field encodings*) which make designing a language for parsing tractable. On the other hand, primitive sensor metadata may suffer from the following inconsistencies which make parsing hard[2] :

- Context-dependence: Different fields may be coalesced in a context-dependent way. For instance, the sixth letter — `C` — in `BLDA1C600A_ART` denotes the value for the field *room*, while in `BLDC1C2____TMR` it denotes the value for field *chiller*.
- Multiple Schemas: Tags within a particular building may comprise several different schemas. For instance, a sensor with the metadata `BLDA1C600A_ART` should be parsed as $\boxed{\text{BLD}}\boxed{\text{A}}\boxed{\text{1}}\boxed{\text{C}}\boxed{\text{600A\_}}\boxed{\text{ART}}$, each token representing the value of a different field. In the same building, there exists sensors with metadata such as `BLDS03AR179ART`, which is tokenized as $\boxed{\text{BLD}}\boxed{\text{S}}\boxed{\text{03A}}\boxed{\text{R}}\boxed{\text{179}}\boxed{\text{ART}}$.
- Variable Delimiters: The metadata schema also does not depend on specific delimiters. As shown in the example $\boxed{\text{BLD}}\boxed{\text{A}}\boxed{\text{1}}\boxed{\text{C}}\boxed{\text{600A\_}}\boxed{\text{ART}}$, letters themselves can be the delimiters for some tokens, and underscore characters for others.
- Spurious Delimiters : Some tokens may have delimiters as a part of the token. For instance, in the same dataset a sensor with the tag `BLDA2S14SASA_M`, for example, should be parsed as $\boxed{\text{BLD}}\boxed{\text{A}}\boxed{\text{2}}\boxed{\text{S}}\boxed{\text{14}}\boxed{\text{SASA\_M}}$.
- Multiple values for the same field: some fields may be expressed by multiple different values. For instance, the field *room* may be denoted by an `R` or a `C`, while the field *damper valve position* maybe expressed as either `VAV` or `VP`.
- Noisy metadata: The tags of some sensors may have misplaced or wrong tokens borne out of human error.

3. AUTOMATED METADATA CONSTRUCTION TECHNIQUES

We now describe the techniques used to perform each of the four main components of our system , i.e apriori Syntactic Clustering (3.1), Rule Synthesis and Application (3.2), selecting an example for the expert using only the sensors' metadata syntax (3.3) or through data (3.4).

3.1 Syntactic Clustering

Given a building's sensor tags, we perform syntactic clustering on them. This preconditioning step has three advantages — (a) tags in a resulting cluster are more regular, and hence helps our rule synthesis algorithm converge , (b) the cluster with the most number of unqualified tags is a good metric to decide which example to next select for an expert's parse , (c) the rule synthesis and application happens only on the tags in the same cluster as the expert-provided example, and is computationally fast.

In constructing the feature vector, we aim to cluster tags which resemble fixed field encodings together. Unlike clustering text documents, we have no apriori notion of delimiters or words. We assume all non-alphanumeric character to be a potential delimiter. We replace contiguous runs

[2]we illustrate examples from one building in our dataset. These challenges appear in all the three buildings in our dataset

of alphabets, numerals and special characters with a single number — alphabets are denoted by `1` , numerals by `2` and each special character as an independent but consistent number. Thus, the tag `BLDA1R465__ART` is denoted in our feature space as $\langle 1, 2, 1, 2, 3, 1\rangle$, and `BLDS03AR179ART` as $\langle 1, 2, 1, 2, 1\rangle$[3]. Intuitively, points which are close together in this space have the same relative positioning of alphanumeric characters, and thus can be parsed by a similar synthesized program. The feature vector corresponding to a particular sensor is then padded with `0`s to make them have the same number of dimensions.

We perform agglomerative clustering based on the jaccard distance between the feature vectors as the distance metric[4]. We define the hetereogeneity metric within a cluster to be the average of all-pair jaccard distances, and at each step merge two clusters if the heterogeneity of the resultant cluster is below a specified threshold[5]. Clusters thus formed are more regular (since they have similar positioning of alphabets, numerals and delimiters). Tags with idiosyncratic schema form their own clusters.

3.2 Rule Synthesis and Rule Application

Our synthesis technique selects a sensor tag and presents the example to an expert for a parse. We first introduce terminology that we will use throughout this section, followed by an overview and a description of the synthesis technique.

Terminology: The expert is expected to point out *(Field, Value, Value Type)* tuples in the sensor tag. A *field* is mapped on to a substring of the tag, which is called its *value*. A field can have a constant or a variable value. A value is a *constant* if it is not specific to that particular tag, and *variable* otherwise.

Sample Input: Suppose the expert is presented with an example `BLDA1R465__ART`. This tag indicates that it is in Building `BLD`, is part of the first air handling unit, indicated by the character `A1`, in room 465 (`R465`) and it is the area temperature sensor (`ART`). The expert should provide the parse as: `BLDA1R465__ART` : (site, `BLD`, const), (ahu, `A`, const), (ahuRef[6], `1`, var), (zone, `R`, const), (zoneRef, `465`, var), (zone air temp sensor, `ART`, const). The *site* field's value is `BLD`, which is not specific to that particular sensor tag. Hence, the expert should mark it as a constant. On the other hand, the value of the *zoneRef* field is specific to that sensor, and hence should be marked as variable.

Sample Output: The synthesis technique should be able to identify the learned fields in a new tag automatically. For example, given a new tag `BLDA5R577A__ART`, it should output the set of tuples: `BLDA5R577A__ART` : (site, `BLD`), (ahu,`A`), (ahuRef,`5`), (zone,`R`), (zoneRef,`577A`), (zone air temp sensor, `ART`, const). If there exists a portion of the tag for which the synthesis technique has not yet received an example, it should remain unmapped to any field-value tuple.

We term each of these tuples as a *qualification*, because it qualifies a set of alphanumeric characters into field-value pairs in a common namespace. A tag is *fully qualified*, if every alphanumeric character in it was correctly *qualified* by the set of outputted *field-value* pairs. The goal of the expert

[3]since BLDA is a continuous run of alphabets, it is replaced by a single `1`

[4]so that strings with a higher number of common coordinates would be clustered together

[5]We set the threshold to 0 in our experiments

[6]ahuRef, zoneRef are idioms from the Haystack taxonomy

should be to use fields from the set of markers and idioms defined in Project Haystack [1]. There might be cases where the correct field (such as specific alarms, etc) is not part of the Haystack taxonomy. In these cases, we expect the expert to use an easily understandable long-form field name, which is consistent[7] across the entire building.

Synthesis technique overview (within each cluster[8]) : The high-level aim of the technique is to learn two sets of information from the given input-output examples — (a) which fields are applicable on a particular sensor tag, and (b) what is the set of regular expressions that transform the tag to the value of the corresponding field.

From each expert example, and for each field in the expert-provided qualification, the set of all expressions from the language (shown in Figure 2), that could extract the required field's value is computed. If there are multiple examples for the same field, the substring extraction rules of the multiple examples are intersected to obtain a more concise set of expressions. If the substring extraction rules cannot be intersected, they are maintained as two disjoint sets, which we shall hereby term as a *partitions*.

Finally, for each field and each disjoint set of extraction rules/regular expressions therein, a classifier in the form of **If Then ... Else** statements is built, where the conditions are boolean in the Disjunctive Normal Form (DNF)[9]. These classifiers dictate whether a particular field is applicable to a particular sensor tag, and which regular expression partition should be applied to it. Thus, we can independently consider each *field* to be a potential output for a sensor tag. If a field is deemed to be applicable by its classifier, then the value of that field would be generated by the regular expressions synthesized by our technique.

Learning a classifier for each field separately has two advantages. The classifier is able to identify and extract fields from other sensor tags, irrespective of the formatting of the rest of the tag. Suppose two zone temperature sensors have the tags `BLDA1R465__ART` (expansion described above) and `BLD_300___ART` : (site, `BLD`), (zoneRef, `300`), (zone air temp sensor, `ART`) . In both cases, the first three characters denotes the value of the *site* field, and the substring `ART` denotes that it was a *zone air temp sensor*. Learning classifiers for the fields *site* and *zone air temp sensor* would enable us to gain useful information by automatically applying these fields on the second sensor, even though we might not know its *zoneRef*. Thus, we are able to transform as much of the metadata as possible without having to depend on another example from the expert.

The language : The language is designed to take into account various possible metadata encodings that can occur in sensor tags. We assume that the substring corresponding to the value of a field can be obtained by either (a) extracting substrings between two constant indices, (b) extracting the substring between two other fields or regular expressions, and (c) as a constant width substring if the left index is identified. The classifiers, which are constructed to determine whether a particular field is applicable on a sensor's tag, is based on the general format of the string — (a) whether a particular regular expression occurs at a particular index, or (b) how many times a particular regular expression occurs in the sensor tag.

Transformation Program **P**	:=	if b_1 then e_1
		else if b_2 then e_2
		…..
		else tag not exist in string
Boolean classifier $\mathbf{b_i}$	:=	$d_1 \vee d_2 \vee \ldots \vee d_n$
Conjunct $\mathbf{d_i}$	:=	$p_1 \wedge p_2 \wedge \ldots. \wedge p_n$
Predicate $\mathbf{p_i}$	:=	**Occurs**(v, r, k) | **OccursAtPos**(v, r, c)
Occurs(v, r, k)	:=	True, iff regular expression `r` occurs in string `v` , `k` times
OccursAtPos (v, r, c)	:=	True, iff regular expression `r` occurs in string `v` at index `c`
Extraction Rule **e**	:=	**Substring** (v, p_1, p_2)
Substring(v , p_1, p_2)	:=	Substring of string `v` between positions p_1 and p_2
Position $\mathbf{p_1}$	:=	**Constant**(k) | **PrecedeSucceed**(r_1, r_2, c)
Position $\mathbf{p_2}$	:=	**Constant**(k) | **PrecedeSucceed**(r_1, r_2, c) | **ConstantWidth**(k)
PrecedeSucceed(r_1, r_2,c)	:=	Index at the cth intersection of regular expression r_1 and r_2
ConstantWidth(k)	:=	Index p_1 + k, where p_1 is starting index of substring
Regular Expression **r**	:=	**Tokens**(T_1, …. T_n)
Token **T**	:=	Alphabets| Numeric | specialToken | ε
		| constant tag value entered by expert

Figure 2: Language for learning substring extraction

The top level expression of the language is the classifier — the *If* b_i *Then* e_i structure, which applies the substring expression e_i to the input only if it matches the boolean expression b_i. The boolean function is in the Disjunctive Normal Form and is composed of predicates of the form **Occurs**(v_i, r, k) , which evaluates to true, iff the input v_i has k occurrences of the regular expression r, or **OccursAtPos**(v_i, r, c) which evaluates to true iff the input v_i has a regular expression r which occurs at index c.

The Substring expression **SubString**(v_i, p_1, p_2), evaluates to the substring between positions p_1 and p_2 of the string v_i. **Constant**(k) denotes the integer position k in the substring. A position expression **PrecedeSucceed**(r_1, r_2, c) when applied on a string s evaluates to an integer position t in the subject string s such that r_1 matches some suffix $s[0..t]$ and r_2 matches some prefix of $s[t...l]$ (where l = Length(s)). Also, t is the cth such match starting from the left end of the string. If such an position t does not exist in the string, this operator fails. **ConstantWidth**(k) is an operator which indicates a constant offset index from the position p_1. The regular expressions are either just a single token τ, or a token sequence, **Tokens**$(\tau_1..\tau_n)$, or ϵ (which matches the empty string). The tokens τ comprise of a single token to denote alphabetic characters (referred to as *AlphTok*) , one for numeric characters (referred to as *NumTok*), one for each special character, and one for each constant value entered by the user. The output is obtained by applying the resultant **SubString**(v_i, p_1, p_2) operation.

We provide a couple of examples to elucidate how the field-value extraction technique works.

Example 1. Tag : `BLDA1R465__ART`, desired output value : `ART` (for the field *zone air temperature sensor*).
Possible programs synthesized: SubString(s, Constant(11), Consant(14)), or
SubString(s, PrecedeSucceed(*UnderscoreToken*, *ART*,1), ConstantWidth(3)) .

Example 2. Suppose the synthesis algorithm has seen two examples (a) `BLDA1R465__ART`, for which the value for field *ahuRef* is `1` and
(b) `BLD_300___ART`, in which the field *ahuRef* does not exist, and hence should not be applied.
Possible programs synthesized to extract the value of the field *zoneRef* is : **If** b_1 **Then** e_1, where b_1 = OccursAtPos(s, (A), 3), e_1 = Substring(s, Constant(4), ConstantWidth(1)).

[7]This can be achieved by presenting the expert a set of fields that has already been used in that building to qualify that particular substring.
[8]the clusters formed in Section 3.1
[9]same technique as in [13]

This program ensures that the field *ahuRef* is only applied to tags similar to the former, and not to the latter.

Example 3. Consider the following tags from our testbed, in which the first sensor is a status indicator connected to supply fan `4`, while the second is a variable air volume unit airflow sensor in room `5871`.

Tag 1 : `BLDA4S1831_STA` : [(site, `BLD`, const), (ahu, `A`, const), (ahuRef,`4`, var), (supply fan,`S`, const), (supply fanRef,`1831`, var), (status point,`STA`,const)] ; and

Tag 2: `BLDA3R5871_VAV` : [(site, `BLD`, const), (ahu, `A`, const), (ahuRef, `3`, var), (zone, `R`, const), (zoneRef, `5871`, var), (vav, `VAV`, const)]

Both these sensor tags have the exact same arrangement of numeric and alphabetic characters, and special symbols, and no classifier comprising only tokens for alphanumeric and special characters would be able to discern between the two. This can result in erroneous extra fields being applied to sensor names.

Token Set: To solve this problem, and get a more expressive set of tokens, we utilize the values marked as *constant* in the examples provided by the expert as special tokens. In the example above, the tag `BLDA4S1831_STA` is treated as a set of tokens — (`BLD`), (`A`), *NumTok*, (`S`), *NumTok*, *NumTok*, *NumTok*, *NumTok* ,*UnderscoreTok*, (`STA`). Note that utilizing the *constant* values as tokens enables us to have a different token set for each building. The set of tokens increase as the expert gives more examples. Note that the new tokens provide enough expressibility for the regular expressions to differentiate between the two tags `BLDA4S1831_STA`, and `BLDA3R5871_VAV`.

3.3 Syntactic Example Selection

We choose the cluster[10] with the maximum number of not-yet-fully qualified sensors, and choose one of them as the next example to present to the expert. We evaluated four different techniques to choose an unqualified sensor from that cluster, the results of which are presented in our evaluation (Section 4.3).

- *Random:* Select an example at random.
- *MinLeft :* Select the example with the minimum tag length left to qualify. The intuition is to complete partial parse of sensors.
- *MaxLeft :* Select the example with the maximum tag length left to qualify. The intuition is to help the synthesis technique cover the space of unseen fields (i.e the long tail in Figure 4).
- *SameLeft :* Select an example with the most frequent unqualified substring. This method seeks to find a commonly occurring field.

3.4 Data-driven Example Selection

Often the same sensor types in a building are specified in the form of multiple different schemas. This may occur because of mistakes on part of the vendor, later addition of such sensors, etc. For instance, zone temperature sensors in Building 1 in our testbed was encoded in 6 disparate schemas, comprising of 78%, 11%, 6%, 2% , 2% and 1% of the sensors. It is unlikely that a purely syntactic example selection method will happen upon the tags with rare encodings. In contrast, because the characteristics in the associated data of sensors of the same type tend to be similar, data-based selection can identify the rare-encoding sensors as being similar to sensors with a more common encoding, and present one of these rare examples to the expert for parsing.

To utilize this approach, an expert should specify which are the sensor types an application needs (henceforth termed as *Required Sensors*). Our technique first transforms each sensor stream into two new data streams M and V that contain the running median and variance values of the original data stream, using a 45-minute long sliding window. The length of the window is set to 45 minutes to smooth over any transient phenomena and noises. We compute the *minimum*, *maximum*, *median* and *variance* of each of the two streams M and V, resulting in the following 8-tuple: $\langle min(M), max(M), median(M), var(M),$ $min(V), max(V), median(V), var(V)\rangle$.[11]

This 8-tuple is then used as a feature vector to train a random forest classifier based on sensors which have already been fully qualified, and to classify the yet-unqualified sensors. We choose a random forest classifier because it is an ensemble learning algorithm which is more robust to noise and in general outperforms other learning techniques like SVM or linear regression. We choose the sensor which the classifier classified as a *required sensor* with maximum likelihood and select that example be parsed by the expert.

3.5 Running Portable Building Applications

We study three important applications, that with normalized metadata can run portably across buildings — Finding Rogue Zones, Finding Zones with Stuck Dampers, and Finding Inefficient Air Handling Units.

1. Finding Rogue Zones : A thermal zone is *rogue* if its air temperature is constantly above its required setpoint, i.e it requires constant cooling. Rogue zones are typically caused by high thermal load, incorrect setpoints, or faulty sensors, and should be rectified before implementing further sophisticated efficiency techniques. Rogue zones are an artifact of poor planning or wrong setpoints, and can be fixed if brought to the attention of a building manager. This application queries for sensors having the *zone air temp sensor* field, and for each such sensor, queries for a sensor with the *zone air temp setpoint* field having the same field-value for the *zoneRef* field, and checks whether the temperature is always more than its respective setpoint (factoring in a tolerance factor of 2F).

2. Finding Stuck Dampers: Finding zones where the dampers are stuck, i.e they do not modulate the amount of chilled air entering a zone. This application queries for all sensors with the field *zone damper* and the corresponding zone number (*zoneRef*), and checks whether its data stream remains constant or shows any variation.

3. Finding Inefficient Air Handling Units (AHUs): An AHU is considered "inefficient" if it serves rogue zones as well as over-cooled zones. Typically, a hot rogue zone drives the AHU to supply air that is too cold, resulting in other zones supplied by the same air handler

[10]as described in Section 3.1

[11]Described in more detail in [14]. In our experiments, we use the same one week time window of data from the month of July for all sensors.

always being too cold and uncomfortable. Identifying such AHUs may lead to making some over-cooled zones more comfortable. This application first computes whether a zone is over-heated or over-cooled using the same technique as the Rogue Zone application, and then queries for *ahuRef* (the air handling unit ID) field of each over-heated rogue zone, and checks if any of the other zones served by the same air handling unit is over-cooled. This application requires the same sensors as the Rogue Zone application, but requires an extra *ahuRef* relationship between the zones.

4. EVALUATION

We first evaluate the advantage of our a priori clustering approach (4.2), and compare convergence with existing spreadsheet synthesis techniques. We next evaluate the effectiveness and convergence of our algorithm for different syntactic example selection mechanisms (4.3). We then compare and contrast the efficacy of the syntactic example selection method (4.4) to the data-driven example selection method (4.5) to parse enough sensor tags to run a particular application, following which we report the results of the portable applications on the buildings in our testbed (4.6).

4.1 Experiment Setup

We manually ground truth-ed all sensor tags in three buildings with BMS' installed by different vendors, having 1586, 2522 and 1865[12] sensors respectively. We simulate the role of the expert in our experiments. When the simulated expert is asked for a parse, it consults the ground truth and provides the correct parse. A sensor tag is considered fully qualified if (a) the correct fields and field-values are extracted by the synthesized rules (b) No extra incorrect field is identified, and (c) the field-values explain every alphanumeric character of the sensor tag.

4.2 Clustering

We evaluate the number of expert examples required for full qualification of all sensors in a building with and without apriori syntactic clustering. With clustering, the synthesized rules from the expert's parsed example are only applied to the cluster the example is in. Without clustering, the synthesized rules are applied to all sensors. We present the results from Building 3 in our dataset. We also compare the results to the parse generated by the spreadsheet synthesis algorithm presented in [13] (Flash-Fill).

Figure 3 shows the rate of full qualification in Building 3. Without clustering there is a sharp drop in the number of sensors fully qualified because erroneous fields start getting applied to sensor tags which were previously fully qualified. Figure 4 provides the intuition for this behavior. In all the buildings, a few fields (about 20 in each building) are applicable on a lot of sensors[13], while there is a long tail of fields applicable only to a handful of sensors. In the absence of clustering, the synthesis technique over-generalizes rules for these rare or erroneous fields. With clustering, 100% of the sensors are fully qualified. The apriori clustering step avoids

[12]we did not have access to sensor stream data for this building

[13]This is pretty common in commercial buildings, where a majority of the sensors are related to zone information. Thus, fields such as *zone temp setpoint*, *zone airflow*, etc are very common

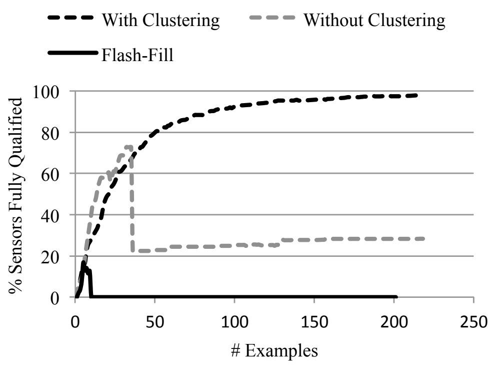


Figure 3: Advantage of apriori clustering in Building 3 : Rules are not over-generalized

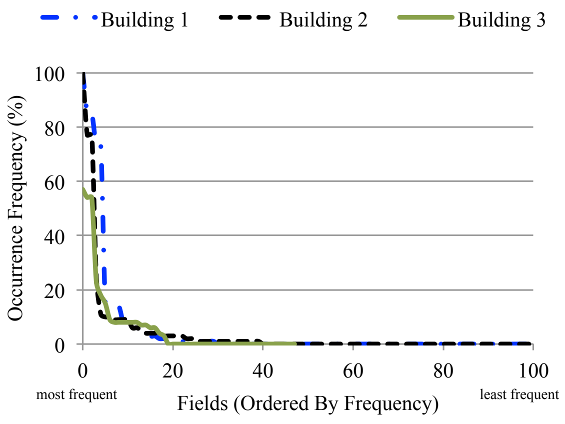


Figure 4: Percentage of sensor names (tags) each field appears in. The x-axis is sorted according to the frequency of occurrence of a field

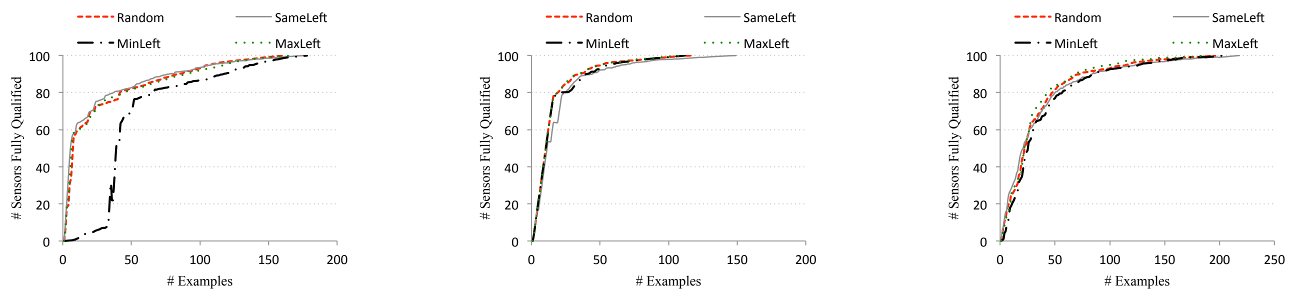
over-generalization of the synthesized rules by restricting the rules to the cluster in which a particular example parsed by the expert lies. The spreadsheet-based synthesis technique (Flash-Fill) fails after 10 examples, because its underlying language and tokens is not robust enough to disambiguate the large number of fields.

Without the apriori clustering, Building 1 also showed a drop in the number of fully qualified sensor tags after 100 examples, settling at full qualification of 90% of the sensors. Building 2's sensors' schemas were much less noisy, and our synthesis technique could parse all sensor tags without clustering (these results are omitted due to space constraints).

4.3 Syntactic Example Selection

We now study the selection methods (described in Section 3.3) used to select which example the expert should parse following the initial clustering.

Figure 5 shows the rate of sensor qualification for each of the four example selection methods on all buildings. With the exception of *MinLeft* in Building 1, all the selection methods qualify sensors at roughly the same rate — they correctly classify the most frequently occurring sensors within a handful of examples. With addition of more examples, the



(a) Building 1

(b) Building 2

(c) Building 3

Figure 5: Sensor qualification rate for the three buildings. The *Random* generator achieves 70% full sensor name qualification within 24 examples for Building 1, 15 examples for Building 2 and 43 examples for Building 3. It takes substantially more examples for Building 3 because its subsystems had no similarity in metadata because they were installed by different vendors.

rate of new sensors fully qualified decrease because the synthesis technique starts encountering obscure fields which are not applicable widely. *MinLeft* performs poorly for Building 1 because its metadata is very noisy, and the approach gets stuck trying to fully qualify idiosyncratic sensors with obscure fields.

Thus, the convergence of the synthesis technique (i.e its ability to fully qualify all sensors) is not affected by decision of the example selection criteria. Also, the apriori clustering enables all the four selection criteria to quickly qualify the most frequently occurring sensors, since all of them seek out examples from the biggest clusters first.

The number of examples required for full qualification of all sensor tags in the three buildings using the *Random* method was 161, 116 and 196 respectively, and that for the *SameLeft* method was 176, 149 and 218 respectively. We use the *SameLeft* method in all the studies in the following subsections because of its deterministic nature.

4.4 Application-Oriented Qualification with Syntactic Example Selection

We present the number of examples required to obtain full-qualification of all sensors for the Rogue Zone application in Building 1 (results for other applications on other buildings are similar, except when stated explicitly). The Rogue Zone application's set of *required sensors* are sensors with the fields *zone temp sensor* or *zone temp setpoint*. In Building 1 there are 462 such sensors, and their tags are in 10 different schemas, some of which are very frequent, while others not so.

Figure 6 shows that 147 examples are required to fully qualify all the *required sensors* for the Rogue Zone application (compared to 176 examples required for full qualification of all sensors). The first two examples fully qualify the *required sensors* (denoted by **P** in the Figure) with the most frequent schemas. *Required sensors* encoded in more obscure schemas with infrequent fields require more examples because the aim of the syntactic selection method is to select examples from large clusters. As the expert parses more examples, the number of fully qualified non-required (**N**) sensors increase steadily. Even though fewer expert examples are required for application-oriented qualification, the number of examples required is still prohibitively large to enable easy deployment of the Rogue Zone application.

The number of examples required to fully qualify all sensors for the Rogue Zone application on Building 2 was 67 (compared to 149 examples for qualifying all sensors). Similarly the Identifying Stuck Dampers application took 137 and 1 example for Buildings 1 and 2 respectively (all dampers in Building 2 were encoded with the same schema).

4.5 Application-Oriented Qualification with Data-Driven Example Selection

We now evaluate the number of examples required to fully qualify sensors for the same application and building as in the previous experiment (i.e Rogue Zone application on Building 1). We use the syntactic selection method to select the first five examples, so that the data-driven classifier has positive and negative instances in its training set, and thereon apply our data-driven classifier, and select the maximum likelihood *required sensor* to present to the expert for parsing.

Figure 7 shows that only 24 expert examples are required to obtain full qualification of all *required sensors*. After step 5, the data-driven classifier has 356 positive and 472 negative examples of fully qualified sensors in its training set. Out of the remaining 758 sensors (its test set, out of which 106 are *required sensors*), it classifies 231 sensors as *required sensors*, out of which 102 are true positive (denoted by **TP** in the Figure). It selects the example which it has classified as *required* with maximum likelihood.

Thus, data-driven example selection (after 5 steps of syntactic example selection) leads to a 6x reduction in the number of expert examples required compared to purely syntactic example selection, making it feasible to deploy the Rogue Zone application on Building 1 within a few minutes.

However, one cannot apply the data-driven technique until the syntactic technique has identified at least one positive example for each required sensor. Figure 8 shows the number of examples needed[14] for full qualification of all *required sensors* for the Rogue Zone and Stuck Dampers application as a function of the initial number of syntactic example selections for Buildings 1 and 2. In general, fewer syntactic steps gave better results. This result was a surprise to us, because our intuition was that our algorithm would need to perform several syntactic steps to build up an adequate training set for the data-driven classifier. However, the *SameLeft* syn-

[14]the sum of data-driven example selection steps and syntactic example selection steps

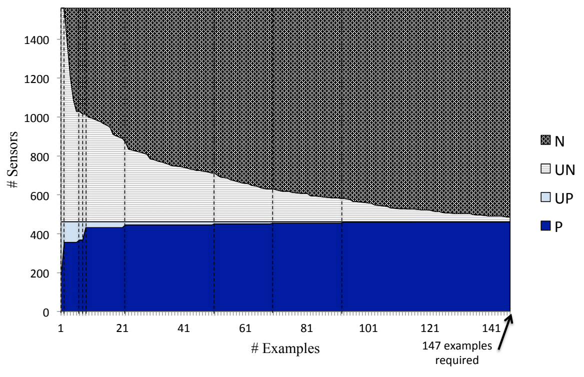


Figure 6: Rate of full qualification of *required sensors* for the Rogue Zone application on Building 1 using *sameLeft* example selection method. (**P**) denotes number of *required sensors* fully qualified (positive), **N** shows number of unrequired sensors fully qualified(negative), **UP** shows the number of *required sensors* yet to fully-qualified (unkown positive), and **UN** shows the number of non-required sensors yet to be fully-qualified (unkown negative). The dotted vertical line shows the steps in which a remaining *required sensor* example was presented to the expert.

tactic selection method always chose the first few example in a way to provide a sufficiently large training set for the data-driven example selection method to progress. Building 2's dampers were all encoded in the same schema, and hence all *required sensors* were parsed by the first syntactic example.

Note that in this experiment, our algorithm stops when the last *required sensor* is qualified. In practice, a heuristic would be required to infer that further positive examples are unlikely to be obtained. Thus, a small number of of additional examples would be required ensure convergence.

4.6 Results of Applications

We ran our portable applications for finding rogue zones, stuck dampers and inefficient AHUs to Buildings 1 and 2 (Table 1), after the expert had parsed all the required sensors.

Results : We were able to identify *all* the zones and dampers in the two buildings. Building 1 was the more inefficient building with 5 hot rogue zones, and 17 over-cooled zones, and 4 zones having stuck dampers. We identified two inefficient air handling units in Building 1 which were trying to cool down extremely hot electrical closets and in the process over-cooling multiple office spaces. The inefficient AHU application could not run on Building 2 because none of the sensors encoded the relationship between zones and their corresponding AHUs.

Table 1: Application Results

	Building 1	Building 2
Number of Thermal Zones	201	78
Number of Rogue Zones	5	2
Number of Over-cooled Zones	17	0
Number of Zones with Dampers	175	55
Number of Zones with Stuck Dampers	5	0
Number of Air Handling Units	4	NA
Number of Inefficient AHUs	2	NA

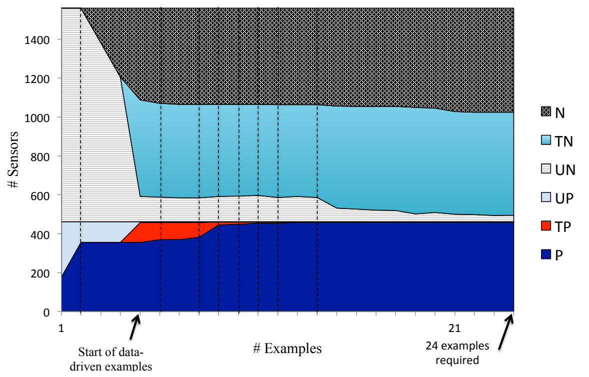
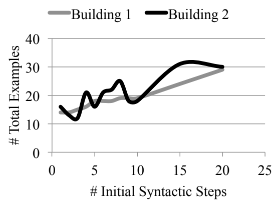
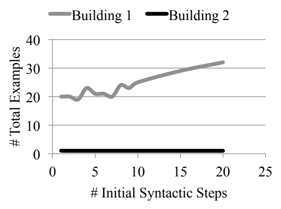


Figure 7: Rate of full qualification for Rogue Zone application in Building 1 using data-driven example selection after 5 steps of syntactic example selection. **P**, **N**, **UN** and **UP** is defined in Figure 6. **TP** indicates true-positive required sensors that have been labelled by the data-driven classifier at that step. **TN** shows the number of true negative *required sensor* identifications made by our data-driven classifier. In this case, **UP** is the same as false negatives and **UN** same as false positives, but this is unknown to the data-driven classifier. The dotted vertical line shows at which step a remaining *required sensor* example was presented to the expert.



(a) Rogue Zone Application



(b) Stuck Dampers Application

Figure 8: Total number of expert examples required to qualify all the *required sensors* for an application, as a function of the number of initial syntactic selection steps (the remaining steps have data-driven example selection).

5. CONCLUSION AND FUTURE WORK

In order to build meaningful applications at scale for buildings with disparate sensor metadata schemas, existing building sensor metadata schemas should be *augmented* and *normalized* to a common namespace to the extent possible. The normalization helps capture the semantic relationships between sensors, which is critical in enabling such applications.

In this paper we developed an approach which can normalize the primitive metadata of each building to field/field-value pairs in a desired common namespace using a few examples from an expert (e.g the facilities manager, often the only person familiar with the existing building metadata). We demonstrated that our synthesis technique is robust and achieves full sensor qualification for all sensors for 3 different BMS systems, even when presented with obscure and noisy tags. Our technique takes very few examples to fully qualify the most commonly occurring sensors (24, 15 and 43 examples for the three buildings in our testbed for qualifying 70% of the tags). We used the relationships inferred in the transformed metadata to run three unmodified analytics applications on two of the buildings. The data-driven example selection method reduces the effort to deploy a new application in an unknown building. The three applications could be deployed in under 30 examples from the building manager.

As part of our future work, we want to perform usability studies with human experts, to devise intuitive ways to enable them navigate through the large building datasets (related research efforts include [3, 18]) and come up with techniques to make our system robust to errors in the expert's parse. Also, we would like to explore if adapting the a priori syntactic clustering of sensors based on examples parsed by the expert reduces the number of examples required.

6. ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation grants CPS-1239552, CNS-0931843 and Intel. We would like to thank Ras Bodik, Emina Torlak, Andrew Krioukov, Stephen Dawson-Haggerty and our shepherd Mario Berges for their helpful feedback.

7. REFERENCES

[1] Project haystack. `http://project-haystack.org/`.

[2] ALC. Automated logic corporation. `http://www.automatedlogic.com/`.

[3] A. Bhattacharya, D. Culler, D. Hong, K. Whitehouse, and J. Ortiz. Writing scalable building efficiency applications using normalized metadata: demo abstract. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pages 196–197. ACM, 2014.

[4] Brown and Caldwell. Scada analysis. `http://sswd.org/modules/showdocument.aspx?documentid=966`.

[5] CitectSCADA. Tagging tutorial. `http://www.citect.com.tw/download/files/1hr_Quickstart_Tutorial.pdf`.

[6] I. F. Classes. Industry foundation classes. `http://www.ifcwiki.org/index.php/Main_Page`.

[7] S. Dawson-Haggerty, X. Jiang, G. Tolle, J. Ortiz, and D. Culler. smap: a simple measurement and actuation profile for physical information. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 197–210, New York, NY, USA, 2010. ACM.

[8] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl. An operating system for the home. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 337–352, San Jose, CA, 2012. USENIX.

[9] D. . V. S. Engineering. Arapahoe county water and wastewater authority electrical, instrumentation and scada system design standards. `http://www.arapahoewater.org/documents/SCADA_Standards.pdf`.

[10] F. Engineering. Scada tagging standards, wastewater treatment division, cincinati. `http://bit.ly/1LvfJQn`.

[11] R. Fontugne, J. Ortiz, and D. Culler. Empirical mode decomposition for intrinsic-relationship extraction in large sensor deployments.

[12] GBXML. Green building xml. `http://www.gbxml.org/`.

[13] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM.

[14] D. Hong, J. Ortiz, A. Bhattacharya, and K. Whitehouse. Sensor-type classification in buildings. *arXiv preprint arXiv:1509.00498*, 2015.

[15] D. Hong, J. Ortiz, K. Whitehouse, and D. Culler. Towards automatic spatial verification of sensor placement in buildings. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, BuildSys'13, pages 13:1–13:8, New York, NY, USA, 2013. ACM.

[16] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011.

[17] M. Koc, B. Akinci, and M. Bergés. Comparison of linear correlation and a statistical dependency measure for inferring spatial relation of temperature sensors in buildings. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pages 152–155. ACM, 2014.

[18] M. Mayer, G. Soares, M. Grechkin, V. Le, M. Marron, O. Polozov, R. Singh, B. Zorn, and S. Gulwani. User interaction models for disambiguation in programming by example. UIST, 2015.

[19] J. Ortiz. *A Platform Architecture for Sensor Data Processing and Verification in Buildings*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2013.

[20] OSIsoft. Pi system. `http://www.osisoft.com/`.

[21] J. Ploennigs, B. Gorman, N. Brady, and A. Schumann. Bead-building energy asset discovery tool for automating smart building analytics: demo abstract. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pages 194–195. ACM, 2014.

[22] A. Schumann, J. Ploennigs, and B. Gorman. Towards automating the deployment of energy saving approaches in buildings. In *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*, pages 164–167. ACM, 2014.

[23] US Department of Energy. 2011 Buildings Energy Book, 2012.

[24] T. Weng, A. Nwokafor, and Y. Agarwal. Buildingdepot 2.0: An integrated management system for building analysis and control. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, BuildSys'13, pages 7:1–7:8, New York, NY, USA, 2013. ACM.

[25] I. Wiese. The integration of scada and corporate it. `http://www.iinet.net.au/~ianw/integration.doc`.