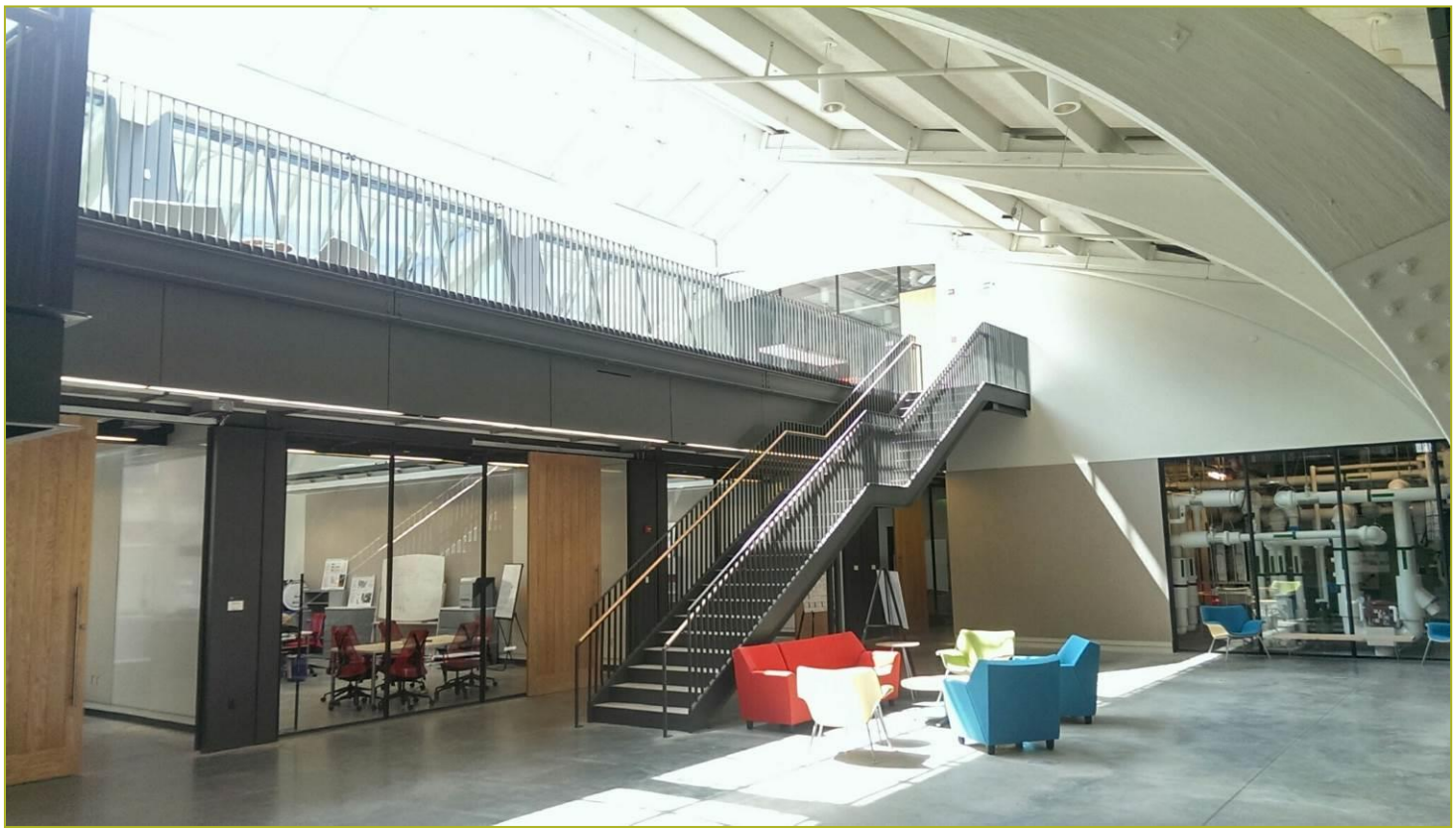
CONSORTIUM for BUILDING ENERGY INNOVATION

REPORT

Title: Volttron application for assisted discovery of semantic meaning of BAS points.

Report Date: April 30, 2016

Report Authors: Francesco Leonardi, Hayden Reeve, Tim Wagner



Funded by U.S. DOE

Report Abstract

The objective of this project is to create a VOLTTRON utility to automatically infer the semantic meaning of building points for cost-effective deployment of intelligent building applications.

During the execution of this project a novel inference algorithm was created. The algorithm was tested against a large and diversified dataset comprising points from five buildings, two vendors, three distributors and more than 20K points. Overall the algorithm identifies about 90% of VAVs and 80% of AHUs and reaches an accuracy of about 90% in detecting the points required by a test application.

Requirements and use cases were collected from market partners that also provided part of the dataset used to develop and test the inference utility.

The algorithm was incorporated in a Python utility that is released to the public as free and open-source. The utility comes with a Graphic User Interface to assist non-expert users in inferring semantics of BAS points. The utility leverages the developed algorithm to automatically infer point semantic and it also allows the user to review and possibly manually correct generated results. The utility is ready to be used within the Volttron framework.

Contact Information for lead researcher

Name: Francesco Leonardi
Institution: United Technologies Research Center
Email address: leonarf@utrc.utc.com
Phone number: +1 860-214-4409

Contributors

Hayden Reeve, United Technologies Research Center

Tim Wagner, United Technologies Research Center

Ziyou Xiong, United Technologies Research Center

Introduction

The primary bottleneck in deploying intelligent applications on top of existing building automation systems is the laborious task of mapping embedded controller's data to application data.

Historically consistent building data-point naming conventions (e.g. BACnet, Project Haystack) have not been used and BACnet semantic tagging standards are still under development. Even when such standards are implemented, there is still the significant challenge of retrofitting existing buildings. Field engineers have the freedom to set up variable names according to their preferences. Typically, names are correlated with the semantic of the variables they represent. This project leverage modern text mining techniques to classify points with high accuracy based on their assumed meaning. The end result is a prototype VOLTTRON utility that supports the deployment of intelligent building applications. This utility drastically reduces the required manual labor required to map points from a BAS to applications.

In the first phases of the project the team reached out the research community (in particular with the Pacific Northwest National Lab, Virginia Tech, and Carnegie Mellon University) in order to synchronize and share opinions on how to tackle the problem and discuss current state of the art. The team also met with market partners (Automated Logic Corporation, Athenianrazak, and Intellimation) to understand their business needs for the assisted inference of BAS points. These market partners can see potential benefits of applying the technology developed in this project in two scenarios: 1) mapping new applications (e.g. reporting, data analytics, supervisory control, diagnostics, etc.) on top of existing BMS/BAS data, and 2) mapping points across various embedded (e.g. BACnet) controllers. It was also suggested that it might be practical to compile a dictionary to facilitate the mapping of points in non-programmable BACnet devices. In fact, these point names are fixed (at least within a product line). It was recommend developing and testing the algorithm against a diversified dataset. In fact, different BMS/BAS have different level of standardization in their point names. According to their opinion, the problem that this project aims to solve is the most critical bottleneck in enabling cost effective analysis of building data.

State of the Art

The last few years witnessed an increasing interested from academia in automatically infer semantic metadata from points. There are three broad approaches on how to semantically label data points. The first approach attempts to augment the information about points by looking at their values and applying techniques to correlate points. Examples of this method are in [1] [2]. The second type of approach, as detailed in [3], [4] [5], and [6] is more similar to the one taken in this project: to infer as much semantic information as possible by just looking at point names. Clearly, the richness and consistency of data points dictates the difficulty of the problem and set an upper bound on the quality of the semantic inference. Finally, the last approach combines both methods to produce more robust and accurate results. The algorithm described in [1] and in [2] learns how to map point names to a prebuild dictionary from a set of examples point names presented by an operator. This algorithm assumes the existence (and the knowledge) of a naming convention of point names. The algorithm attempts to match each

point with a definition in the dictionary. The richer the dictionary and its definitions the easier it is to find the right match. An alternative technique is presented in [3]. It iteratively learns the naming schema of point names and associates this schema to a common semantic model like Haystack [4]. Similar to the previous approach the algorithm generates questions (or examples) that a user needs to answer. A characteristic of this solution is that the operator has to provide the "translation" between the naming schema and the semantic model. Therefore this approach is well suited for buildings with limited naming schemas. The problem of "transfer learning" i.e. the ability to learn a model in one or few building and to reuse the same model across several buildings to classify/label points has been addressed only recently. A promising approach is presented in [5] where the algorithm combines both sensor readings and point names to learn a classifier that can be applied to other buildings. While results are encouraging in labelling point types there is no attempt in inferring relationships between points. Similar conclusions can be drawn from [6] where an active learning algorithm can achieve very high accuracy in establishing point types. In the same paper it also noted that for many practical uses, identifying point types is not sufficient. Invaluable information such as identifying equipment types and grouping points per equipment can lead to a drastic simplification in the deployment of building applications. The algorithm developed within this project attempts precisely to tackle this last problem.

Workload Reduction

In order to truly understand the impact of any tool for semantic inference of BAS points, a cost model for mapping data points from a BAS into an application must be introduced. Assuming that an operator requires a constant amount of time to assign each semantic tag, a simplified cost model can be written as a function of the number of tags (τ) that need to be assigned:

$$\varphi(\tau) = \|B\|\tau_B + \sum_{b\in B}\sum_{e\in E} I(e,b)\left(\tau_e + \mathrm{P(e)}\tau_\mathrm{p}\right)$$

Where B represents a set of buildings, $\tau_\mathrm{B}$is the number of tags that are specific to a building (examples of building tags are name, location, area, type, climate zone, etc.), E is the set of equipment types that are of interest of the application, function $\mathrm{I(e,b)}$ returns the number of instances of an equipment type e in a building b, τ_e is the number of tag per equipment type (examples of equipment tags are type, location, relationship with zones and other equipment, etc.), function $\mathrm{P(e)}$ returns the number of points in equipment type e, and τ_p is the number of tags per point (examples of point tags are: point definition and reference to containing equipment). The first term of the cost model ($\|\mathrm{B}\|\tau_\mathrm{B}$) is usually small compared to the second one. In fact in a typical campus there are many more equipment and points than buildings and $\tau_\mathrm{B}$is similar to τ_e.

Practical Example

Introducing an example can help picturing the potential benefits of the use an assisted inference utility like the one created in this project. This example consists of a fictitious campus comprising 30 buildings (i.e. $\|B\| = 30$). An energy-management contractor needs to perform some data analysis on all VAVs in the campus. This particular application requires 5 tags per building ($\tau_\mathrm{B} = 5$). In this example there are

on average 100 VAVs per building and that the application analyses data streams of 4 points per VAV. Each VAV must have at least one tag ($\tau_e = 1$) to describe the type of equipment, so that the application can recognize it. Given an average of 3 tags per point and 4 points per VAV, then $\mathrm{P(e)\tau_p} = 12$. With the current technology the energy-management contractor has no choice but to manually assign about 40000 tags. Even assuming a generous 30 seconds per tag, this requires about 300 hours to complete the job. The utility developed in this project can automatically identify about 90% of the tags needed and therefore can bring down the number of hours of manual work to about 30.

Algorithm description

In its current version the algorithm digests a list of BAS point names as input and generates suggestions of the likely meaning of points as output. Future versions might use additional knowledge about points (e.g. values, measurement units, etc.) to increase the robustness and the accuracy of the output. The meanings of points are expressed by a combination of an internally defined set of semantic tags. A user

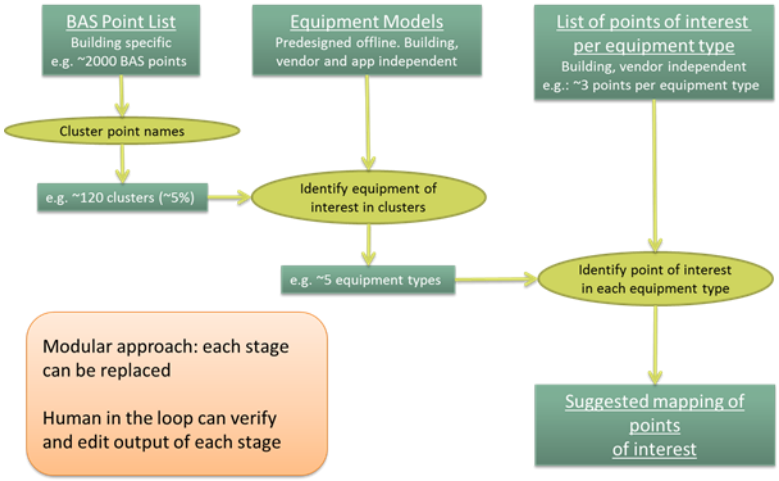


Figure 1: Block diagram of the inference algorithm

of this application has the ability to replace these tags with industry standard ones (e.g. Haystack) and/or extend with custom ones.

The inference algorithm can be logically divided into three steps: point clustering, equipment identification, and point identification (Figure 1). The algorithm takes in input a list of point names from a building automation system, a set of equipment models, and a list of point types of interest per equipment type.

Points clustering

In this unsupervised step the algorithm first divides each point name into its constitutive parts (i.e. tokens) and it constructs a graph representation of the tokenized points. Within this graph the algorithm searches for nodes that are "similar." Nodes that are similar to each other are placed in the same cluster. Two nodes are considered "similar" when most of their children have the same token. The reasoning behind this algorithm is that similar nodes are likely to belong to the same equipment type

(e.g. VAVs in a building are likely to have more or less the same point names encoded in similar naming conventions) or other logical groups (e.g. list of equipment, subcomponent of equipment etc.). It has been observed that this algorithm can usually identify between 20 and 100 clusters in commercial buildings.

Table 1: Dataset composition and cluster results.

Building		#BAS points	#C	#Unique points in clusters	#Un-clustered points
ID	Vendor				
1	JCI	1160	26	126 (10%)	0
2	Siemens	4308	36	145 (3.3%)	121 (2.8%)
3	JCI	3545	50	329 (9.3%)	1
4	ALC	5015	93	432 (8.6%)	200 (4.0)
5	Siemens	1808	29	101 (5.6%)	82 (4.5%)
6	Siemens	5981	49	213(3.6%)	119(2.0%)
7	Siemens	8247	38	205 (2.5%)	321 (3.9%)
8	ALC	3448	44	211 (6.1%)	53 (1.5%)
9	ALC	3428	33	143 (4.2%)	85 (2.5%)

Equipment identification

The second step of the inference algorithm attempts to automatically assigning a semantic meaning to the clusters created in the previous step. In particular the current version of this algorithm is interested in establishing whether a cluster represents HVAC equipment and the label of the equipment (e.g. VAV, AHU, FCU, etc.). In order to achieve this goal a Support Vector Machine classifier has been trained specifically to identify VAVs and AHUs. Other classifiers have been considered, but the SVM offers the best performance in this particular problem. The second input of the algorithm (i.e. the set of equipment models) represents the support vectors used for online classification.

Point identification

The final step in the algorithm consists of identifying points of the equipment matching the point types of interest (third input of the algorithm). A point type is characterized in a formal semantic (e.g. Haystack tags, custom tags, etc.) and it carries a set of tokens that are commonly used in the field for that point type. To this end the algorithm computes the distances between points of the equipment and the point types. Then it assigns the type with the lowest distance to the equipment point.

Algorithm Performance

This section summarizes the performance of the algorithm on a large dataset comprising data points from 9 buildings that uses BAS from different vendors and dealers.

Table 2: Equipment identification results.

ID	#VAV			#AHU		
	GT	FP	FN	GT	FP	FN
1	107	0	0	8	0	1
2	51	1	1	-	-	-
3	148	0	1	6	0	0
4	101	0	0	7	2	0
5	181	0	16	6	0	0
6	82	0	5	-	-	-
7	421	3	0	15	3	0
8	52	3	0	2	0	0
9	100	0	0	1	0	1

Points Clustering

Table 1 summarizes the point clustering results. The first three columns report the building ID, vendor name, and the number of points in the building. Column 4 lists the number of concepts obtained by the algorithm. Column five represents the number of unique points in the clusters. The last column summarizes the number of points that the algorithm is not able to assign to any cluster. The number of points per building ranges from about 1000 to 8000. The dataset is divided in a training set (building 1 to 4) and a test set (building 5 to 9). The first one used to create library models and tune parameters. The second one serves to assess the algorithm performance. The test set consists of data points collected from 5 buildings with 2 BMS vendors and 3 dealers. The proposed algorithm identifies about 20 and 100 clusters per building. More importantly the number of points in the semantic graph $\tilde{G}$ is in the order of about 3 to 7% of the total number of points and the number of un-clustered points is less than 5% of the total number of points. This means that by assigning a semantic definition to each cluster and to each cluster-point it is possible to annotate most of the dataset. In other words clustering by itself can drastically reduce the effort required to label a dataset.

Equipment Identification

Equipment identification results are shown in Table 2. The first columns represents the building ID, columns 2 and 5 report the ground truth (GT) of the number of VAV and AHU per building. Columns 3 and 4 summarize the number of false positives (FP) and false negatives (FN) of VAV classified by the algorithm. Columns 6 and 7 list similar results for the AHUs. The algorithm accurately recognizes more than 90% of the VAVs and more than 80% of the AHUs present in a building. Worth to mention the classifier is unable to detect the only AHU of building 9. This happens because of a current limitation of the clustering algorithm that requires the presence of at least two similar groups of points/equipment in order to create a cluster. In this case, having only one AHU, the clustering algorithm does not create a cluster for the AHU points and therefore the classification algorithm is unaware of it.

Table 3: Point Classification VAV.

ID	Sns Dmpr Position			Sns Flow Supply Air			Air Flow Supply SP			Sns Zone Temp		
	GT	FP	FN	GT	FP	FN	GT	FP	FN	GT	FP	FN
1	107	0	0	107	0	0	-	-	-	107	0	0
2	51	0	1	51	0	1	51	0	1	51	1	1
3	148	0	1	148	0	1	148	0	1	144	-	5
4	101	0	0	101	0	0	101	0	0	4	0	0
5	-	-	-	181	0	16	-	-	-	181	0	16
6	82	0	5	82	0	5	82	0	5	82	0	5
7	421	3	0	421	3	0	421	3	0	421	3	0
8	52	3	0	52	3	0	52	3	0	21	0	0
9	-	-	-	100	0	0	100	0	0	98	0	0

Table 4: Point Classification AHU.

ID	Sns Rtrn Air Temp			Sns Mixed Air Temp			Outside Air Dmpr			Sns Rtrn Air Hum			Sns Outside Air Temp		
	GT	FP	FN	GT	FP	FN	GT	FP	FN	GT	FP	FN	GT	FP	FN
1	8	0	0	8	0	0	8	0	0	8	0	0	8	0	0
2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	2	-	-	6	0	0	6	0	0	2	4	0	-	-	-
4	7	2	0	7	2	0	7	0	0	7	-	-	-	-	-
5	6	0	0	-	-	-	-	-	-	-	-	-	6	0	0
6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	15	0	0	15	0	0	-	-	-	15	0	0	-	-	-
8	2	0	0	2	0	0	-	-	-	1	0	1	-	1	-
9	1	0	1	1	0	1	1	0	1	1	0	1	-	-	-

Point Identification

Point identification results for VAVs and AHUs are reported in table 3 and 4 respectively. The first raw of both tables indicates some of the supported point types expressed with custom semantic tags. For each point type three columns summarize the number of occurrences of the point (GT), the number of false positives (FP) and false negatives (FN) identified by the algorithm. In most cases, the algorithm classifies point with more than 90% accuracy. Similarly as for the equipment identification, building 9 represents an exception. In fact, all the AHU points (for building 9) cannot be detected. The point identification algorithm operates on labeled concepts. Since the concept for the AHU in building 9 is not even generated, its points cannot be discovered.

Software Architecture

The architecture of this software application is essentially composed of a back-end and a front-end. The first one implements the algorithms and the data structures needed for the execution of the application.

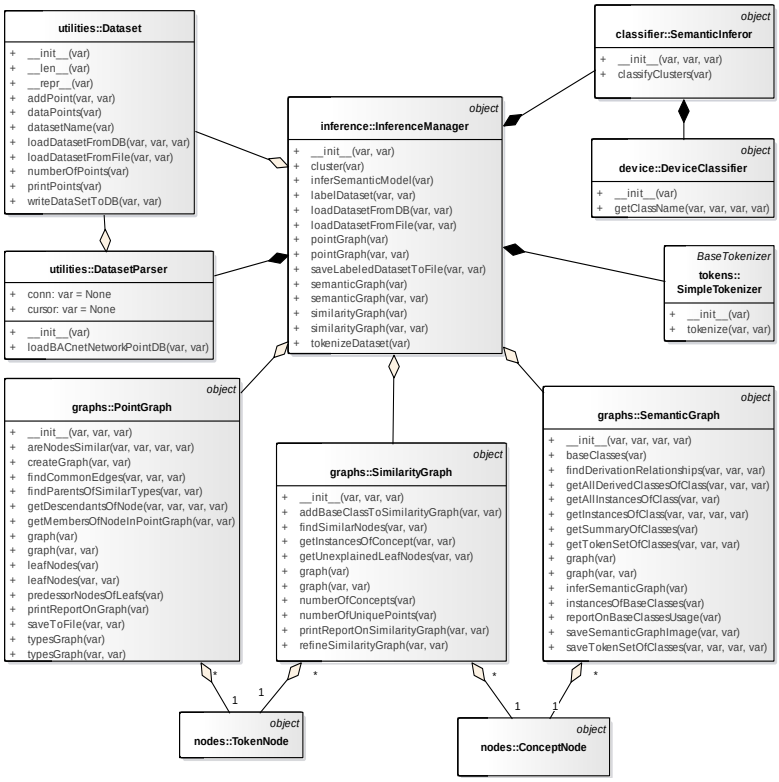


Figure 2: Class diagram of the back-end

The front-end is responsible creating the interface between the operator and the application.

Back-End

The back-end class diagram1 is depicted in **Error! Reference source not found.**. Its components are described in more details in the following paragraph.

InferenceManager: This component provides a high level interface to perform macro activities. It is also responsible of creating the data model shared among the various components of the back-end. More specifically it implements these functionalities:

- Dataset loading: either from a file or from a database. This is accomplished by instantiating a "DatasetParser" and properly invoking its methods
- Tokenization of a point list: This is eventually delegated to an instance of a "SimpleTokenizer"

[1] A class diagram describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects [18].

- Dataset clustering: This uses instances of PointGraph, of SimilarityGraph, and SemanticGraph. The first of these two objects is responsible for the creation of a graph of tokens. Each node in this graph is a "TokenNode" object. The instance of the SimilarityGraph identifies clusters of tokens (in the graph of tokens). For each cluster a ConceptNode object is created and the relationship between each ConceptNode and tokens is maintained. Dataset clustering also extracts a SemanticGraph from the other two graphs.
- Cluster classification: This delegates the responsibilities of labelling ConceptNodes in the SemanticGraph to the SemanticInferor.
- Dataset labeling: labels the entire dataset, based on the information present in each ConceptNode. This information is either generated by the application (automatic Cluster classification) or provided by the operator.

DatasetParser: This component hides the complexity in loading a dataset from various sources (currently both files and databases are supported) and in populating an internal data-structure to maintain the dataset in memory.

BaseTokenizer: A base class that provides the basic functionality to tokenize each point of a dataset. Sub-classes must implement the "tokenize" method that divides a point name into tokens.

SimpleTokenizer: This class derives from the BaseTokenizer class. It implements the logic of how to obtain a set of tokens from a point name. In this current version whenever it encounters special characters ("#", "-", "_", ".", ":", "/", " ") or a transition from lower to upper case in the point name, it divides the input string into tokens.

PointGraph: This class contains a graph and supporting tooling to populate it and to operate on it. Once the graph is populated it is intended to offer a graphic representation of the tokenized point names. Each node on the graph is a TokenNode.

TokenNode: This class maintains a relationship between a token and the point from which the token was derived.

SimilarityGraph: This class contains a graph and supporting tooling to populate and to operate on. The graph contains two types of nodes. ConceptNodes that represent "concepts" of which clusters of tokens to offer a graph representation of the tokenized point names.

ConceptNode: Instances of this class represent clusters of points. Clusters are collections of "similar" points. Within the scope of this application each cluster is assumed to define a concept whose meaning is assigned either by the application or by an operator.

SemanticGraph: This class contains a graph and supporting tools to populate and to operate on. The graph contains ConceptNodes and it captures the relationship among ConceptNodes that can be derived by looking at points.

SemanticInferor: This module oversees the classification operations. It introduces some semantic reasoning to increase robustness of the results and to attempt classifying clusters. When this is not sufficient it calls the DeviceClassifier to provide a lower-level label assignment. It also assigns labels to each point in a ConceptNode.

DeviceClassifier: A factory that returns a classifier that attempts to classify ConceptNode (i.e. cluster of points) into equipment.

SVMClassifier: The online Support Vector Machine classifier that is trained to classify clusters into equipment. Currently it can assign labels to VAV and AHU.

Front-End (or Graphic User Interface)

The Graphic User Interface follows a Model Controller View [7] architectural design pattern. The GUI is based on PySide [8] (a Python binding for QT [9]). The look-and-feel of the interface was designed in "QT Designer". The GUI mainly consists of the following classes:

- MainController: Implements the logic to react to user actions. Depending on the user action and on the state of the application, it commands the InferenceManager to execute certain functionalities.
- MainView: Contains the logic to capture user interaction with the GUI.
- DataModel: Consists of a proxy to access instances of the SimilarityGraph, SemanticGraph, and PointGraph in the "back-end" of the application and a cache of other data displayed on the GUI.
- App: The entry point for the GUI version of this application. It loads all the GUI (generated by the QT Designer) and instantiates the model, the view, and the controller of the GUI.

A few other classes are omitted from this report for brevity because they have minor importance (e.g. custom widgets).

Third Party Software Dependencies

The application described in this document is written in Python 2.7 and it depends on the following third party software libraries:

- Jellyfish (version 0.5.1) [10]: Used to compute the "Jaro-Winkler" distance between strings.
- Networkx (version 1.9.1) [11]: From this library the application imports only the direct graph class and it uses only methods to add nodes and to get predecessors and successors of nodes.
- Psycopg2 (version 2.6.1) [12]: Used to create a database connection to load point names (in the case they are stored in a database).
- Sklearn (version 0.15.2) [13]: Used to learn a Support Vector Machine model to classify clusters into equipment and to execute the online classification.
- Python logging [14]: Used throughout the code to log various information about the execution of the application.
- PySide (version 1.2.4) [8]: Used to implement the GUI in QT.

Graphic User Interface

This section describes the interactions between an operator using the GUI and the software components presented in the previous section. The main goal of a user of the software application described in this report is to semantically tag (or label) data points in a building automation system. To achieve this end goal the user needs to proceed through a sequence of steps that are required by the current algorithm. This section introduces five of those steps that summarize the common use of the application by a field engineer. Each of these steps corresponds to a use case[2] and is illustrated by a sequence diagram[3] and a screenshot of the GUI. More specifically the steps are: 1) dataset loading, 2) dataset tokenization, 3) dataset clustering, 4) dataset classification and 5) dataset labeling.

Dataset loading

This is the use case when a field engineer/operator wants to load a dataset (i.e. a set of point names) into the application. In the main panel of the GUI (see Figure 4) the operator can direct the application to load a dataset either from a file, from a database, and in future versions, from BACnet. By clicking on the corresponding button the operator triggers all the operations shown in the sequence diagram of Figure 3. After a few seconds (depending on the size of the dataset), all points are visualized on the main panel.

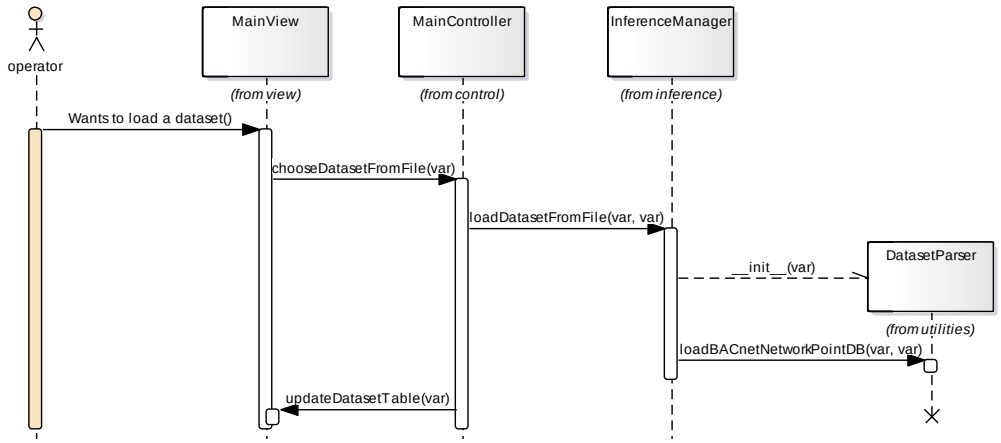


Figure 3: Sequence diagram summarizing the flow of operations triggered to load a dataset from a file.

[2] In software and systems engineering, a use case is a list of actions or event steps, typically defining the interactions between an actor (e.g. a user) and a system, to achieve a goal**Invalid source specified.**

[3] A sequence diagram shows object interactions arranged in time sequence. It depicts the objects involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario**Invalid source specified.**

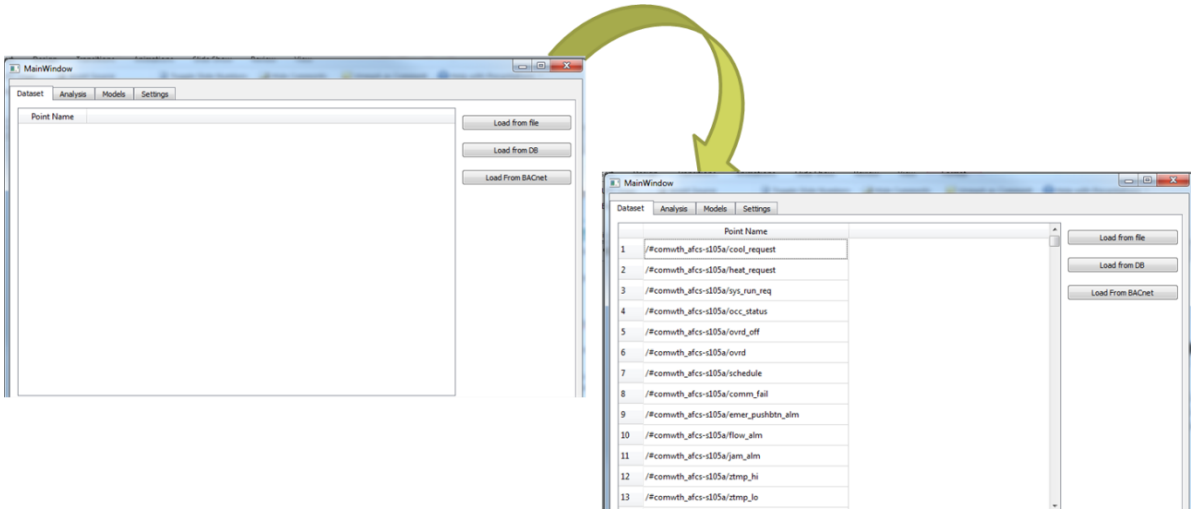


Figure 4: Main panel of the GUI. Before and after points are loaded into the application

Dataset tokenization

This second use case is when the operator wants to tokenize a dataset that is already loaded in memory (i.e. at the completion of the previous step). This step is important because it provide some insight on the "quality" and complexity of the dataset (e.g. too many tokens with low frequency could indicate poor compliance to standards and conventions). By entering the "Analysis" panel (Figure 6) of the GUI, the operator can direct the application to tokenize the dataset. Once all operations are completed, all tokenized points are listed in a table. Information such as number of tokens and frequency of tokens is also reported.

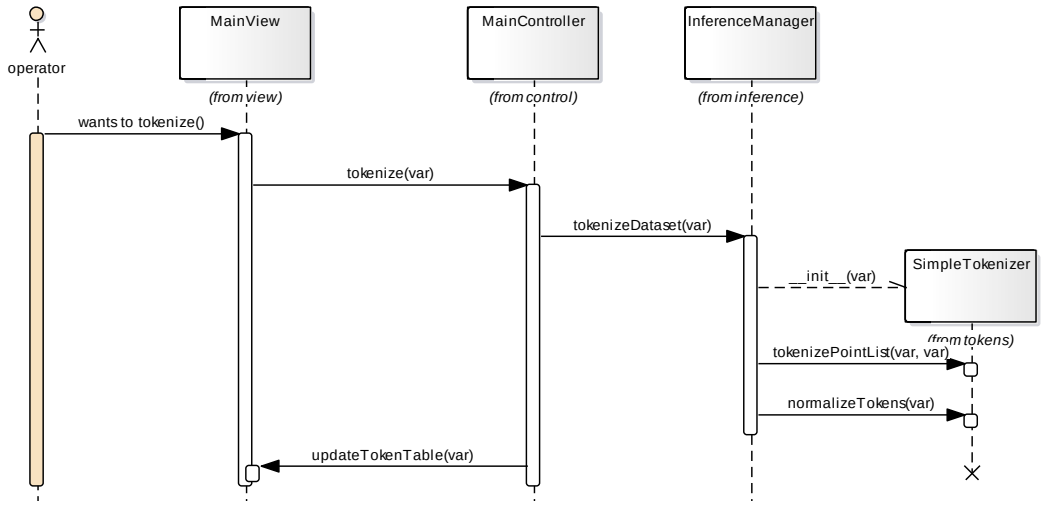


Figure 5: Sequence diagram summarizing the flow of operations triggered to tokenize a dataset

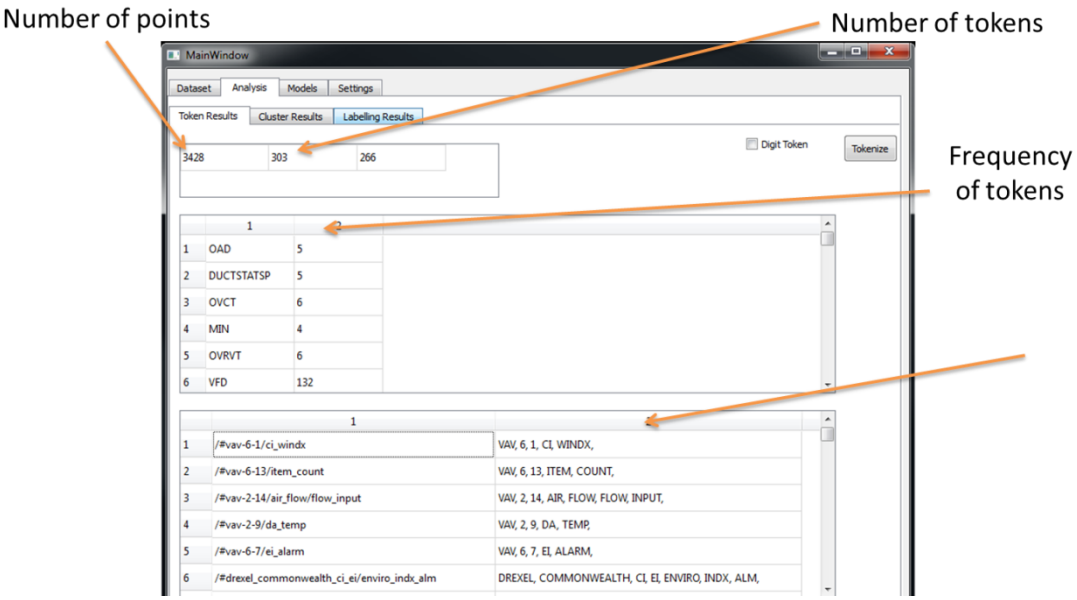


Figure 6: Panel showing a tokenized dataset

Dataset clustering

In this third scenario the operator wants to cluster the dataset. Once the button “cluster” in the “Analysis” panel is clicked, the sequence of actions depicted in the sequence diagram of Figure 7 initiates. At completion, the “Cluster Result” panel is updated (Figure 8). This panel first summarizes the number of clusters found and the number of unique points in all clusters. Then it lists all clusters and it reports the following fields: “ClusterID”, “class”, number of instances, and points per cluster. The first field is a tentative name assigned to the cluster that does not necessarily carry any semantic information with it. The second field “class” is a place holder that eventually will contain the type of equipment (if any) that the cluster represents. The last two fields are useful to quickly identify possible equipment among the clusters (e.g. a cluster with either too few or too many points is unlikely representative of equipment). Each cluster can be expanded to visualize its points (Figure 10). Finally, at the bottom of the panel (with the yellow background) are listed the points that the algorithm was not able to assign to any cluster.

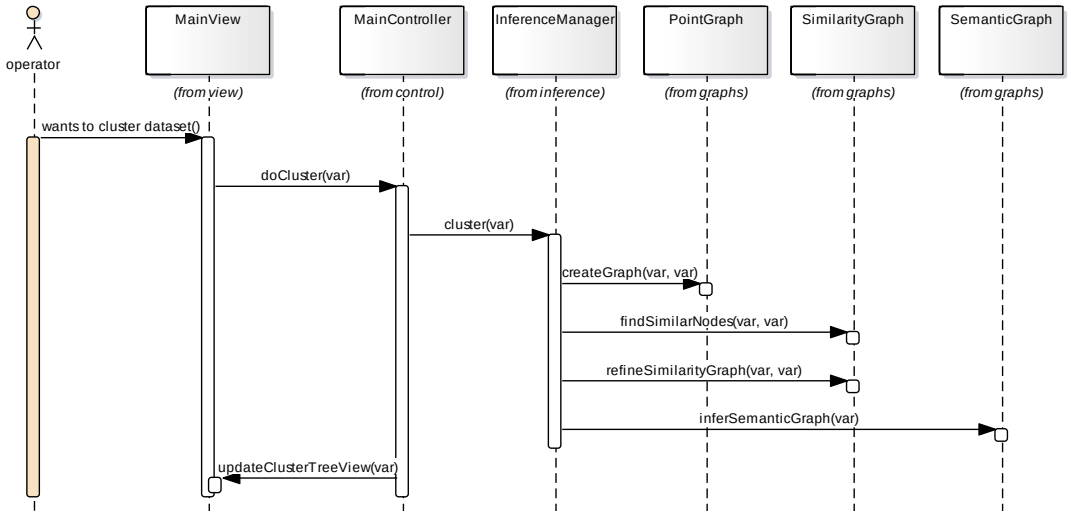


Figure 7: Sequence diagram summarizing the flow of operations triggered to cluster a dataset

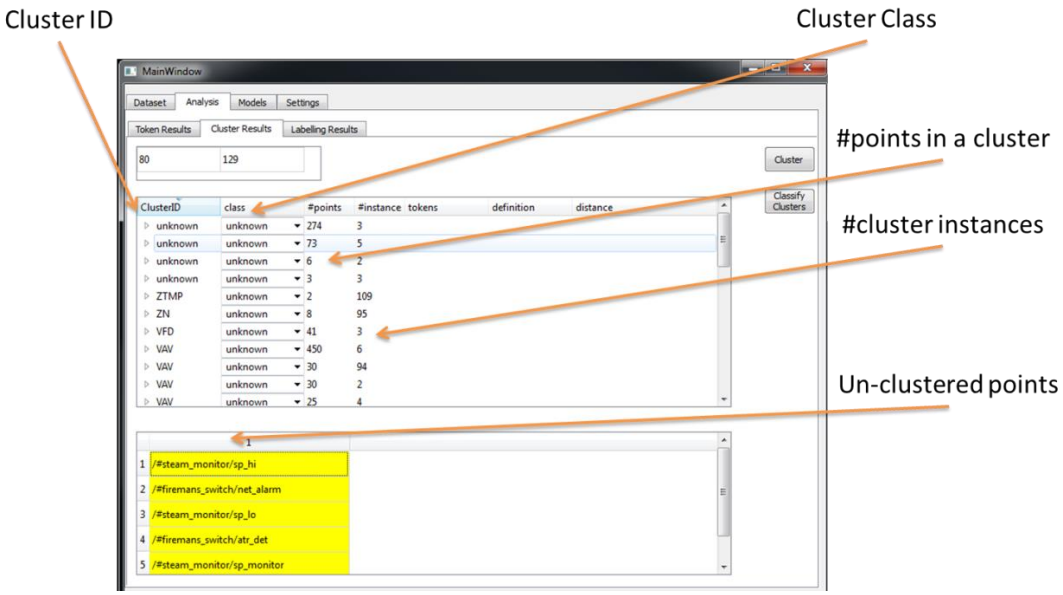


Figure 8: Panel showing a clustered dataset

Dataset classification

In this use scenario the operator wants to attempt to automatically assign class labels to the clusters (i.e. identify which clusters represent equipment). From the "Cluster Results" panel the user can begin this automatic process by clicking on the "Classify Clusters" button. The sequence of operation is shown in Figure 9. At the end, the table summarizing the clusters (Figure 10) is updated showing the computed cluster labels. By expanding each cluster, it is possible to visualize the point labels that the algorithm assigned. Each point label has a score associated with it reporting the likelihood of a correct assignment. It is worth noting is that the operator can override both cluster and point labels.

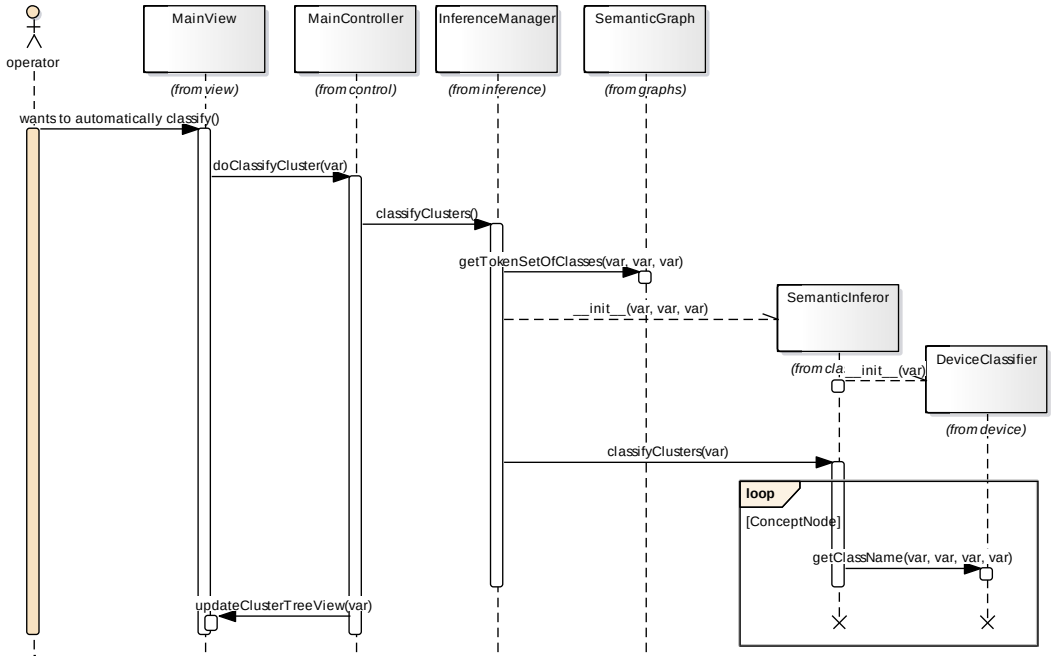


Figure 9: Sequence diagram summarizing the flow of operations triggered to classify a dataset

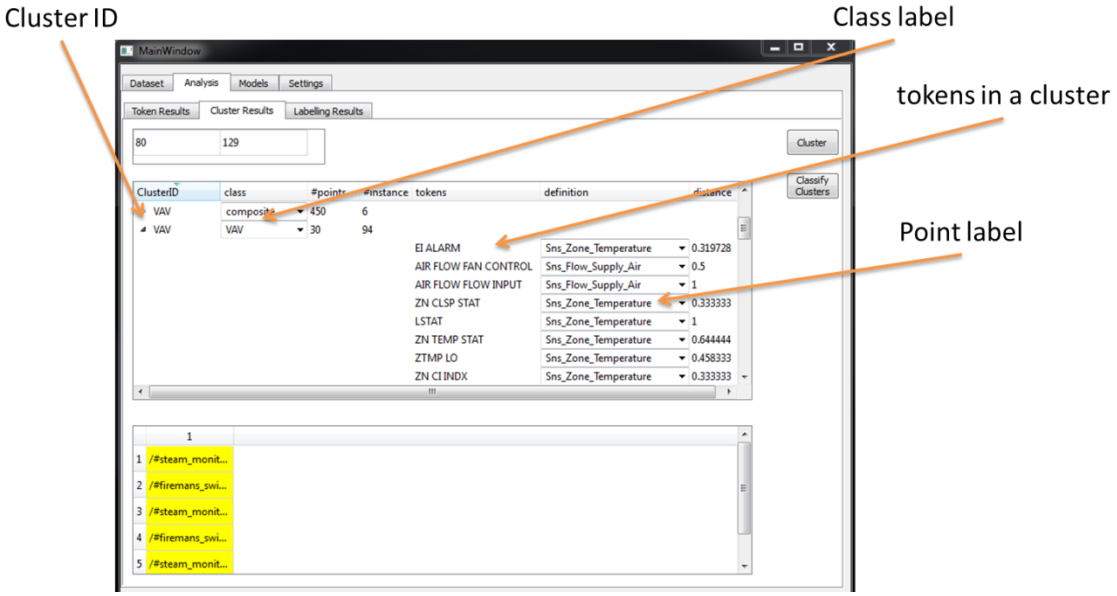


Figure 10: Panel showing a classified dataset

Dataset labelling

The last scenario covers the case when the operator intends to see the labeled dataset. The operator needs to switch to the "Labeling Result" panel (**Error! Reference source not found.**) and click on the "Label" button. Within a few seconds, the panel shows a list of points with semantic tags describing the equipment to which the points belong and other properties.

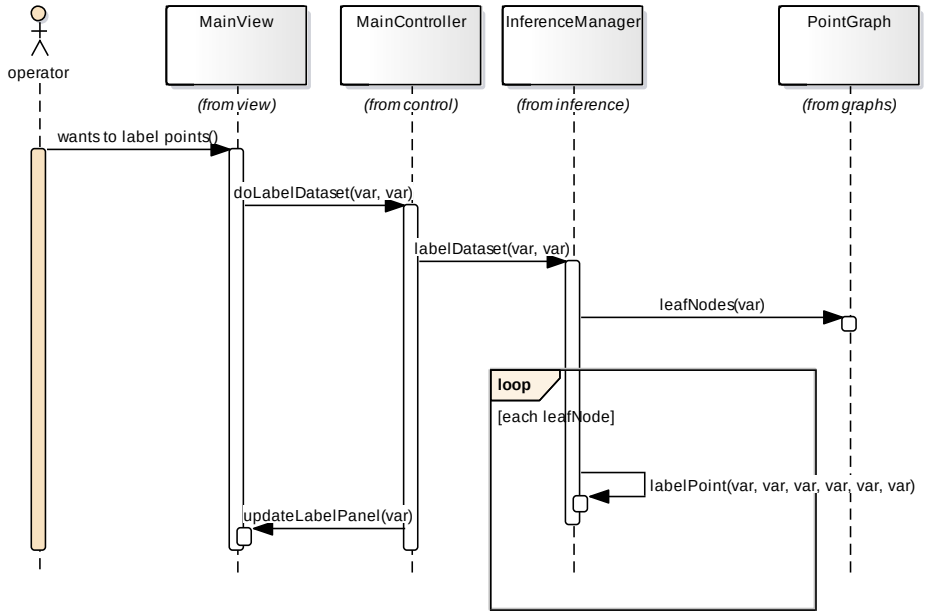


Figure 11: Sequence diagram summarizing the flow of operations triggered to label a dataset

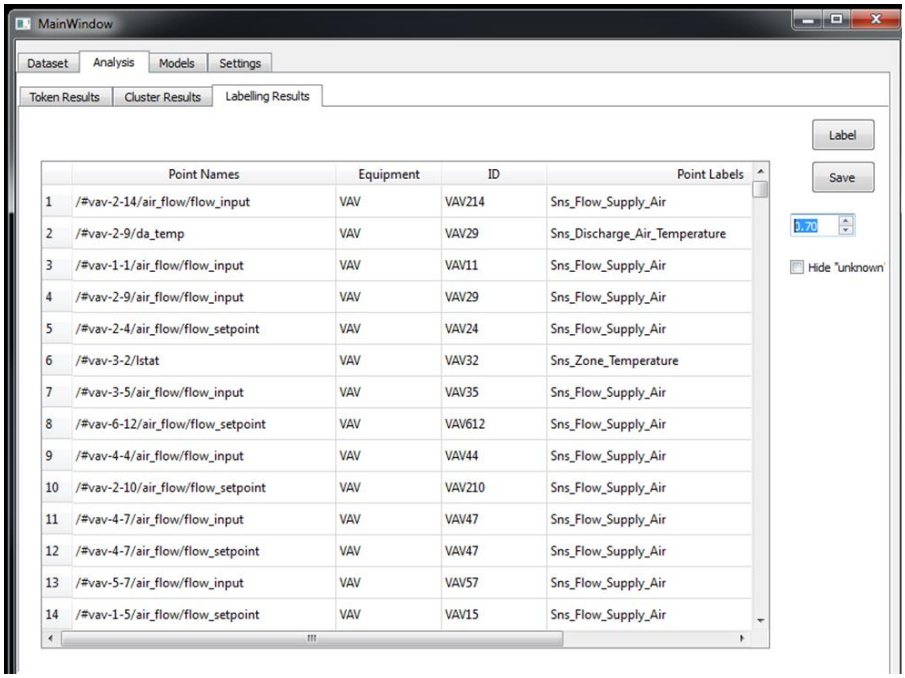


Figure 12: Panel showing a labeled dataset

Acknowledgments

The project team would like to express its special gratitude to Terry Herr president of Intellimation for his help in collecting data points from various buildings.

Bibliography

[1] 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*, 2014.

[2] A. Schumann, J. Ploennigs and a. 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*, 2014.

[3] A. Bhattacharya, D. E. Culler, J. Ortiz, D. Hong and K. Whitehouse, "Enabling portable building applications through automated metadata transformation," University of California, University of

California, 2014.

[4] "Project Haystack," [Online]. Available: http://project-haystack.org/. [Accessed 26 04 2016].

[5] D. Hong, H. Wang, J. Ortiz and K. Whitehouse, "The Building Adapter: Towards Quickly Applying Building Analytics at Scale," in *Proceedings of the 2nd ACM Conference on Embedded Systems for Energy-Efficient Buildings*, 2015.

[6] B. Balajiy, C. Vermay, B. Narayanaswamyy and Y. Agarwal, "Zodiac: Organizing Large Deployment of Sensors to Create Reusable Applications for Buildings," in *Proceedings of the 2nd ACM Conference on Embedded Systems for Energy-Efficient Buildings*, 2015.

[7] [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.

[8] [Online]. Available: https://pypi.python.org/pypi/PySide/1.2.4.

[9] [Online]. Available: http://qt-project.org/.

[10] [Online]. Available: https://pypi.python.org/pypi/jellyfish.

[11] [Online]. Available: https://networkx.github.io/.

[12] [Online]. Available: https://pypi.python.org/pypi/psycopg2.

[13] [Online]. Available: http://scikit-learn.org/.

[14] [Online]. Available: https://docs.python.org/2/library/logging.html.

[15] J. Ortiz, H. Dezhi, 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*, 2013.

[16] M. Koc, B. Akinci and M. Berges, "Comparison of linear correlation and a statistical dependency measure for inferring spatial relation of temperature sensors in buildings," in *Proceeding BuildSys '14 Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings* , 2014.

[17] A. Schumann and F. Lécué, "Minimizing User Involvement for Accurate Ontology Matching Problems," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[18] [Online]. Available: https://en.wikipedia.org/wiki/Class_diagram.

[19] A. Bhattacharya, J. Ploennigs and D. Culler, "Short Paper: Analyzing Metadata Schemas for Buildings: The Good, the Bad, and the Ugly," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, 2015.

[20] M. Pritoni, A. A. Bhattacharya, D. Culler and M. Modera, "Short Paper: A Method for Discovering Functional Relationships Between Air Handling Units and Variable-Air-Volume Boxes From Sensor Data," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments* , 2015.