OpenBAN: An Open Building ANalytics Middleware for Smart Buildings

Pandarasamy Arjunan[†], Mani B. Srivastava[§], Amarjeet Singh[†], Pushpendra Singh[†]

[†]Indraprastha Institute of Information Technology
Delhi, India
{pandarasamya,amarjeet,psingh}@iiitd.ac.in

[§]University of California
Los Angeles, United States
{mbs}@ucla.edu

Abstract

Towards the realization of smart building applications, buildings are increasingly instrumented with diverse sensors and actuators. These sensors generate large volumes of data which can be analyzed for optimizing building operations. Many building energy management tasks such as energy forecasting, disaggregation, among others require complex analytics leveraging collected sensor data. While several standalone and cloud-based systems for archiving, sharing and visualizing sensor data have emerged, their support for analyzing sensor data streams is primitive and limited to rule-based actions based on thresholds and simple aggregation functions. We develop OpenBAN, an open source sensor data analytics middleware for buildings, to make analytics an integral component of modern smart building applications. OpenBAN provides a framework of extensible sensor data processing elements for identifying various building context, which different applications can leverage. We validate the capabilities of OpenBAN by developing three representative real-world applications which are deployed in our test-bed buildings: (i) household energy disaggregation, (ii) detection of sprinkler usage from water meter data, and (iii) electricity demand forecasting. We also provide a preliminary system performance analysis of OpenBAN when deployed in the cloud and locally within a building.

Categories and Subject Descriptors:
H.4 [**Information Systems Applications**]: Miscellaneous
D.2.11 [**Software Engineering**]: Software Architectures

General Terms: Design, Architecture, Deployment

Keywords: Middleware, Building Management System, Smart Buildings, Context Aware Applications

1. INTRODUCTION

Buildings are an attractive target for using novel Cyber-Physical control systems to achieve energy sustainability goals [1]. They are increasingly instrumented with many subsystems that use heterogeneous sensors and actuators for monitoring, automating, and optimizing different building operations [2, 3]. Sensors in buildings generate a large number of data streams that are processed by different energy management applications for control actions. With increasing availability and affordability of sophisticated sensing, control and computational methods, a variety of novel applications have been proposed in the recent past. Example applications include occupancy based lighting, heating and cooling control system for reducing energy wastage [4, 5, 6]; energy forecasting for demand response [7]; energy disaggregation for providing appliance level energy consumption feedback [8], and activity recognition [9].

Despite their potential utility, these novel building energy management applications are still not in widespread usage today because, (i) these applications are mostly prototyped as isolated implementations using off-line data processing tools, (ii) current legacy building management systems often do not provide adequate support for integrating new building management applications, and (iii) it is non-trivial to replicate an application in a setting which is different for which it was originally designed. Modern building management systems, proposed by the research communities, provide rich and sophisticated support for communicating with sensors and actuators, and for archiving, sharing, and visualizing the data [10, 11, 12, 13]. However, the support for processing sensory data is far less mature, mostly limited to rule-based actions performed on the basis of simple thresholds and ranges.

Realization of many of the novel energy management applications requires continuous computation of context information from sensor data. For example, "whenever the meeting room is unoccupied, turn off the air conditioner". The ability to generate and utilize such context information is absent from most of the existing building management systems. To support the computation of context information about building operations from distributed sensory data streams, we propose OpenBAN, a sensor data analytics middleware for buildings. OpenBAN architecture is designed to scale across local and cloud-based deployments, and to support diverse array of services and platforms designed for networked sensors. It provides a runtime environment for developing and scheduling *Contextlet* — a pipeline of processing elements for inferring a particular building context from sensory data. Furthermore, OpenBAN is designed to enable building facilities department to connect various building sensor data streams with different existing and new control applications through a powerful analytics engine capable of inferring context information. Moreover, OpenBAN facil-

.

MOBIQUITOUS 2015, July 22-24, Coimbra, Portugal

DOI 10.4108/eai.22-7-2015.2260256

itates inclusion of several analytical algorithms as part of the *sense-analyze-act* pipeline for developing novel building energy management applications.

Using our prototype implementation, we developed three concrete applications to demonstrate the utility of OpenBAN for a range of applications based on our testbed buildings: (1) disaggregating household appliance usage; (2) identifying sprinkler usage violation from water meter data, and (3) forecasting hourly energy usage from smart meter data.

The primary contributions of our work are as follows:

- We propose OpenBAN middleware architecture that facilitates integrating *distributed sensor data streams* to infer rich context information about building operations using different machine learning algorithms.
- We validate the suitability of OpenBAN by deploying it for three real-world energy management applications in our test-bed buildings.
- We release an open source implementation of OpenBAN containing several analytical algorithms and a host of features pertinent to building applications.

Rest of the paper is organized as follows. In Section 2, we present the background details and motivation. We then present the system architectural details in Section 3. In Section 4, we present the implementation details of the system. In Section 5, we present the details of three representative energy management applications developed using OpenBAN and preliminary performance analysis. We present the related work in Section 6. Section 7 outlines the future work followed by Section 8 concludes the paper.

2. BACKGROUND AND MOTIVATION

Modern commercial buildings often employ a Building Management System (BMS) for managing their complex operations. A BMS consists of many subsystems including, HVAC for indoor climate control; lighting systems for ambient brightness; access control systems for security, and smart meters (electricity, water and gas) for monitoring utility consumption. Sensors and actuators in these subsystems are networked using standard protocols such as BACnet[1] and connected with their corresponding control applications through BMS software. Generally, a BMS is installed with a fixed number of closed-loop control applications which are accessible only by the facilities department.

Similarly, residential buildings are increasingly instrumented with Home Automation Systems (HAS). Several sensors (e.g., temperature, light intensity, motion, door contact status, and smart meters) and actuators (e.g., plugs, door locks, and dimmers) are connected to a HAS controller using wireless protocols such as Z-Wave. The HAS controller box contains a gateway for connecting it to the Internet. Unlike BMS software stack, HAS controllers are flexible and extensible. Some HAS controllers, such as HomeSeer[2], provide SDKs for developers to write home automation plug-ins and allow sharing among different home owners. Home owners install and configure the home automation applications using the provided web interface and/or mobile application.

2.1 Evolution of building middleware systems

Diverse sensor data streams generated in HAS and BMS subsystems are often discarded after they are processed by

[1] http://www.bacnet.org
[2] http://www.homeseer.com

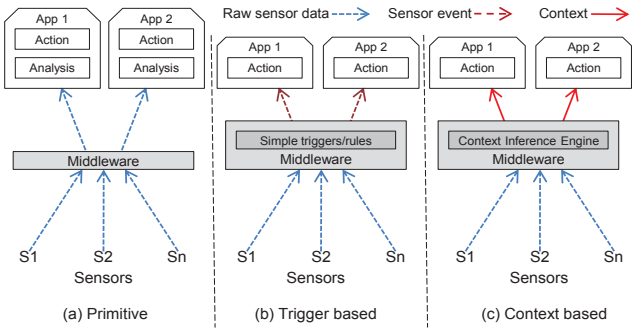

Figure 1: Evolution of building middleware systems based on support for processing sensor data. (a) primitive middleware provides no support for sensor data analytics, (b) rule-based middleware provides trigger-actions based on thresholds, and (c) context-based middleware provides sophisticated analytics for inferring context from sensory data.

their corresponding applications. Since these sensor data streams encompass several operational characteristics of a building (e.g., occupancy information, energy usage patterns), storing and thereafter analyzing them can help optimize building operations. Motivated by this, recent efforts have sought to redesign the legacy building management systems for not only storing such rich sensory data, but, also enabling novel applications to process them. Examples of such research systems are HomeOS [10], SensorAct [11], BuildingDepot [13], and BOSS [12]. Most of these systems provide new abstractions, in the form of RESTful APIs or RPCs, for accessing the underlying distributed network of sensors, actuators, and their data. In addition to this, these systems provide an *application runtime environment* for developing and executing novel applications that can access and process the sensory data [10, 11, 13, 14].

Such redesigned middleware systems for buildings, have opened up opportunities for both researchers and developers to experiment complex yet futuristic building control applications such as personalized HVAC control [14], occupancy prediction [5], and energy monitoring [15]. Although these applications are relatively complex, their processing pipeline is still similar to existing commercial BMS and HAS applications: access raw sensory data, process it and perform the desired control actions. While these redesigned middleware systems provide abstractions for accessing sensors and their data, their inherent support for processing sensory data within the middleware is limited. Based on support for processing sensor data streams, existing middleware systems proposed can be broadly classified into two categories: *primitive* and *trigger-based*, as illustrated in Figure 1.

Primitive support middleware systems expose only the raw sensory data to building management applications. The applications can query historical data or subscribe to real time sensory events. It is the responsibility of the applications to process the sensor data stream in accordance to the desired application. Applications often match the queried raw sensory data with some threshold value for identifying a particular building context. As an example, *AppDoorNotifier* security application in HomeOS detects abnormal activity by comparing the timestamp of door sensor events with a predefined time period. Recent research systems such as HomeOS [10], SensorAct [11] and BuildingDepot [13] are examples of primitive support category.

Trigger-based middleware systems, on the other hand, al-

Table 1: Motivating energy management applications which require complex features and analytics on top of the collected sensor data.

Application	Example features	Machine learning algorithm
Energy disaggregation	Power, Difference in successive power readings, Raw voltage and harmonics, Current, Power factor	Combinatorial Optimization
Sprinkler usage violation	Mean, standard deviation, range and time of the day	Support Vector Machine (SVM)
Energy forecasting	Mean temperature, mean humidity, time of the day, day of the week, day of the month	Logistic regression

low applications to inject triggers for monitoring sensory events. These triggers involve simple threshold based conditions applied over raw or aggregated data, corresponding to identifying a particular building context. Middleware system monitors sensory data events and invokes the applications whenever the trigger condition is satisfied. Unlike *primitive support* middleware systems, identifying the occurrence of a building context is done by the middleware and applications are responsible for executing only the actions. Most of the home automation systems support trigger-based actions. For example, Vera[3] supports trigger-based action schemes such as "when the temperature is below a threshold, turn on the thermostat".

Most of the existing building control applications are performing relatively simple operations. However, novel applications proposed recently involve complex sensor data processing methods. Often these applications apply computationally intensive machine learning algorithms for detecting complex building contexts, such as activity monitoring [9] and occupancy detection [5,6]. We argue that such essential yet complex sensor data analyzing functions should be an integral component of the middleware system instead of implementing them in each application separately. Moving such complex algorithms into the middleware not only makes the applications lighter but also provides better abstractions for accessing them, as shown in Figure 1. Further, common contextual information can be computed centrally and shared with multiple applications. Motivated by this, we seek to design a sensor data analytics middleware, called OpenBAN, that provides a runtime environment for developing and scheduling complex context identification algorithms.

2.2 Motivating Applications

We now present the details of three motivating energy management applications which require context of the building operation. Our aim is to understand the requirements and goals of a middleware system enabling building applications involving complex computation.

Energy disaggregation: Previous studies have shown that providing appliance level consumption feedback to consumers can help them save upto 15 % energy [16]. However, instrumenting individual appliances with power monitors to infer detailed appliance consumption information can be expensive, difficult to maintain and is considered intrusive. In contrast, Non Intrusive Appliance Load Monitoring (NILM) [8] or energy disaggregation is a viable method for identifying individual appliance level usage from the household total power meter readings. A typical NILM setup involves complex pattern recognition algorithms to disaggregate total consumption (as measured at the meter level) into constituent appliances consumption.

Inferring sprinkler usage policy violation: Sprinklers for irrigating lawns, plants, and flower-beds are quite common in many households across the world. Since sprinklers consume large amounts of water (for example, a single sprinkler station may consume as much as 70 cc per second), most cities impose restrictions on when and for how long can sprinklers be used. For example, Los Angeles Department of Water and Power (LADWP) has a policy according to which *"Spray head sprinklers are allowed up to 8 minutes per watering station per day. They are restricted to hours before 9:00 a.m. and after 4:00 p.m."* [17]. While the law exists, the compliance is not good and enforcement nonexistent. We designed an application for detecting sprinkler usage compliance with the help of the water usage trace of an entire house, as recorded by the water meter.

Energy forecasting: Buildings are one of the largest consumers of electricity, accounting for 71% of the overall usage in USA [18]. Governments and utilities across the world have proposed different strategies for optimizing the energy efficiency of buildings. To assess the efficacy of such strategies, various models for predicting future electricity demand have been studied in the past. One such strategy, well known as demand response (DR), focuses on modifying the building energy consumption as per the demand on the grid [7]. To facilitate assessment of such energy efficiency policies, we developed an energy forecasting application that predicts the next hour electricity usage based on historical trends.

Table 1 summarizes the statistical features and machine learning capabilities which these three applications require. From our own experience with ad hoc implementation of these applications, we concluded to design OpenBAN to provide centralized and reusable analytical support for all these and similar complex building energy management applications. Further, many of these applications require an inference output on a continuous basis and thus, this is also a requirement which OpenBAN adheres to.

3. SYSTEM ARCHITECTURE

Based on the specification of our motivating applications discussed in Section 2.2, Table 2 lists the functional requirements of OpenBAN and its corresponding architectural components that meets the identified requirements. As shown in Figure 2, OpenBAN middleware consists of four major components: (i) *Data adapters*, (ii) *Feature repository*, (iii) *Model repository*, and (iv) *Context Inference Engine*.

3.1 Data Adapters

Data adapters are connectors that enable OpenBAN to act as an interface between diverse *Sensor Data Services* for receiving sensor data and sending out the context inferences. It can be either pull-based or push-based. Sensors can also be directly connected with OpenBAN if they are accessible through web APIs. The primary responsibility of these data adapters is to handle the interoperability and data exchange issues across diverse sensors, actuators, applications and their corresponding communication interfaces and APIs.

An *Input data adapter* retrieves sensor data from a particular service and converts it into a common format (<timestamp,value>) understandable by OpenBAN. Whereas, an

[3] http://getvera.com

Table 2: List of system requirements for designing an analytics middleware and the corresponding system components.

System requirements	Architectural elements	Description
Integration of existing and new sensor data streams	Data adapters	Provision to integrate internal and external sensor data services. Support for pull and push based sensors.
Extracting features from sensor data streams	Feature repository	A repository of commonly used features from sensor data analysis literature. Provision for adding additional features and reusing them.
Support for including various analytics algorithms	Model repository	A repository of commonly used analytics algorithms and trained models. Provision for adding additional features and reusing them.
Experimenting and deploying the analytics application pipeline	Context Inference Engine (CIE) and a scheduler	An execution environment for scheduling the analytics algorithms. Provision to scale up the computing power.

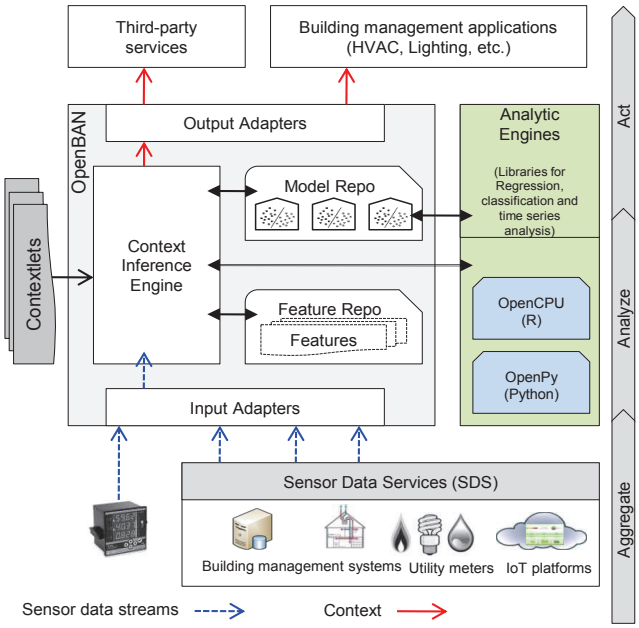

Figure 2: OpenBAN architecture shows the *Aggregate-Analyze-Act* pipeline of building management application. It integrates multiple sensor data streams, and computes the required context information in real time which can be used by multiple energy management applications.

Output data adapter converts the computed result (context inferences) into a format required by the target service and sends it to the target service. Data adapters for existing building subsystems, and external sensor data services, such as *GreenButton*[4] and Xively[5], are implemented and integrated with OpenBAN as plug-ins.

Therefore, any building sensor data service with a compatible data adapter can be integrated with OpenBAN. Such integration of multiple sensor data services into a single framework enables the user to create a custom application pipeline (in OpenBAN) that receives data from multiple sensor data streams, computes relevant inferences, and then communicates the learned inferences to external services.

3.2 Feature Repository

Once a *Data adapter* is available for a sensor stream, the next step is to derive a set of suitable features from raw sensor data values for further analysis, e.g. the average electricity consumption in an hour. In order to facilitate the *feature identification* step as a part of the "sense-analyze-act" application pipeline, OpenBAN provides a *Feature Repository* containing a candidate set of *feature names* which are commonly used in the literature for processing sensor data streams. Each *feature name* in the repository is associated with a *feature function* that computes the corresponding *feature value* over the specified *time slice window*. A time-series sensor data stream is split into a sequence of contiguous time windows at each N-second interval (feature window size). Thus, each time window contains a tuple of $<timestamp, value>$ pairs. The feature function receives a time window as a parameter and returns the computed feature value for each of the time windows.

These feature functions are enclosed with a piece of code written in a high-level programming language, such as Python, using various mathematical and statistical library functions. Such shallow association of a feature name with a piece of program code allows users to easily contribute additional and reusable features to the repository. OpenBAN provides an interface for users to easily create additional features. Once a new feature is created and made available to the repository, other users can also use that feature for their analysis. In this way, OpenBAN creates an ecosystem wherein domain experts, researchers and application developers can collaborate, reuse and share features with others. Table 3 lists a candidate set of different categories of features that were derived from various sensor data streams to infer a range of context information in buildings. These commonly used feature names were populated from literature [5, 9, 19]. Several additional features can also be computed from these basic features e.g. ratios of different quantities.

In addition to providing support for computing features from the Feature Repository, OpenBAN also facilitates creating *Feature templates*. A feature template is an abstraction over a set of features derived from a particular sensor data stream, which are required to infer a specific building context information. Essentially, it is represented as a triplet of {*Context-inference-name, Sensor-name,* {*Feature-name1, ..., Feature-nameN*}}. As an example, a feature template for inferring binary occupancy information from electricity meter readings could be {*Binary-occupancy,* {*Electricity-meter-power, mean, standard-deviation, range*}} [20]. Similar to feature names, feature templates for a specific application can also be created and shared with other users.

3.3 Model Repository

OpenBAN allows users to experiment and include several analytics algorithms as an integral component of the *sense-analyze-act* application pipeline. In order to facilitate complex analytical computations by non-experts, OpenBAN provides a *Model Repository*. It provides a set of analytical algorithms and their model instances that are

[4] http://www.greenbuttondata.org
[5] http://xively.com

Table 3: List of different categories of features that can be identified from various sensor data streams to infer a wide range of context information of a building. These features are computed for each *time window (Tw)* spanning a N-seconds interval.

Feature	Description with example usage	Sensors	Context and applications
Statistical features		Temperature, Motion, CO_2 level, Light-intensity, Door status, Electricity meter, Water meter, Gas meter, Wi-Fi status, Network traffic, Security and access control, and RFID tag data	Occupancy sensing (presence, count, identity, location, and prediction), Energy data disaggregation, Energy (electricity, water and gas) usage prediction and forecast, Load shedding for demand-response, Activity monitoring, and anomaly detection
`min (Tw)`	Minimum light intensity level of a room		
`max (Tw)`	Maximum temperature of a workspace		
`mean (Tw)`	Hourly average electricity usage		
`median (Tw)`	Median CO_2 level in an hour window		
`sum (Tw)`	Total water consumption of a day		
`count (Tw)`	Number of times door closed/opened		
`range (Tw)`	Temperate range of a workspace in a day		
`mode (Tw)`	Mode of a list of active network ports		
`stddev (Tw)`	Standard deviation of gas usage		
`var (Tw)`	Variance in the power usage		
Temporal features			
`time-of-day`	Time of the day e.g morning and evening		
`hour-of-day`	Hour of the day		
`day-of-week`	Day of the week		
`day-of-month`	Day of the month		
`week-day`	Is't a week day?		
`week-end`	Is't a week-end?		

commonly used for inferring various context information, as listed in Table 3, from different sensor data streams in a building. There are four categories of analytics algorithms in the repository: 1) Regression analysis, 2) Classification, 3) Time series analysis, and 4) User contributed algorithms. Similar to *Feature repository*, each analytical algorithm in the repository is associated with some meta information, such as model description and parameters, and a link for invoking the corresponding analytics function. This link points to a HTTP API endpoint in an Analytics Engine (see Section 3.4) that hosts and executes the corresponding analytics function. Users can integrate these algorithms easily, to infer a particular building context with the help of a user interface (see Section 4.4). Additional analytics functions, developed by OpenBAN users, can be made accessible to other users by updating the repository with the required meta information.

3.4 Analytics Engine

Sensor data analytics applications typically require significant computing power as they inherently use complex optimization functions. Analytics engines are dedicated external systems in the OpenBAN architecture, as shown in Figure 2, that provide required computing power (CPU and memory) for executing various analytics library functions in real time. Analytics Engine is designed as a separate entity to ensure system scalability as it manages a large number of sensor data streams and draws the necessary inferences from each of them. Moreover, such loosely coupled design allows for easy integration of additional analytics engines or provision of multiple running instances of an analytics engine which are hosted within buildings or in cloud.

In addition to providing a platform for executing the analytics functions in real time, an analytics engine may also provide persistent trained models. A *handle* of the persistent trained model and its meta information can be updated into the *Model repository* (see Section 3.3) for later use. Optionally, these trained models can also be exchanged with other analytics platforms using Predictive Model Markup Language[6] standard.

[6] `http://www.dmg.org/v4-1/GeneralStructure.html`

3.5 Context Inference Engine

Context Inference Engine (CIE) in OpenBAN is the overall coordinator of the system. Based on analytics application requirements, it wires up other components in a sequence and creates an execution pipeline through which sensor data streams flow. The CIE can be executed in two modes: 1) *Training mode* to learn model parameters for a specific building context, and 2) *Execution mode* to execute a previously learned model over live sensor data streams. Figure 3 illustrates the workflow of both the training mode and the execution mode.

3.5.1 Training mode

Many novel building management applications require learning a model about a specific context of their interest. Typically, learning a model involves learning the correlation between the sensor data and the occurrence of the desired context events over a period of time e.g. learning an occupancy model using motion sensor data labeled with ground truth occupancy patterns. OpenBAN provides integrated support for training these models. The required parameters to train a model are: (a) a list of sensor data streams and their corresponding ground truth labels for the training period, (b) a list of feature names from the *Feature repository* for each sensor data stream and feature window interval, and (c) the model algorithm. Based on these parameters, CIE performs the following steps:

1. Fetches sensor data streams for the given training period using the corresponding data adapters.
2. Splits the time-series data, for each sensor, into a sequence of time windows and invokes the specified *feature function* for each time window to compute the corresponding feature vector.
3. Combines the feature vector with the ground truth labels to create a training data set by aligning the timestamps.
4. Invokes the corresponding API for learning the model, provided by an *analytics engine*, given the training set.

After successfully learning the model parameters a handle for the model is returned by the *analytics engine*. This handle and the meta information about the learned model are stored in the model repository for later use during the *execution mode*.

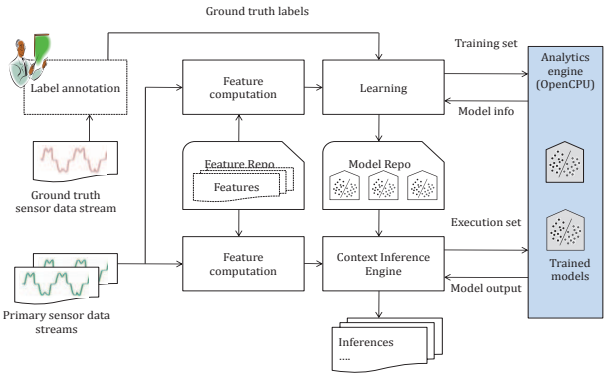

Figure 3: The workflow of the Context Inference Engine for training and execution mode.

3.5.2 *Execution mode*

A user can integrate the previously trained model (or a shared trained model from other users) into their analytics application pipeline. In the execution mode, previously trained models are executed over live or historic sensor data streams based on user specified intervals and time schedule, e.g. "execute the occupancy prediction model every 2 minutes between 9 a.m to 7 p.m. everyday". For each execution instance of such a model, CIE fetches sensor data and computes the required features, as explained in steps 1-2 for the training mode, and creates an execution dataset. Thereafter, CIE invokes the corresponding API function provided by the *analytics engine* with the execution dataset and a *handle* to the previously trained model. The computed results returned by the analytics engine can be sent to multiple external services, based on application preferences.

4. IMPLEMENTATION

In this section we describe the details of a prototype implementation of OpenBAN architecture, as described in Section 3. We have leveraged several open source technologies for our implementation, and have released the code as open source[7]. We have used Java-based Play framework[8] to implement the data adapters, feature and model repository, context inference engine, and a sample user interface for OpenBAN. The Context Inference Engine described in Section 3.5 uses Quartz scheduler[9] to schedule and execute the previously learned models.

4.1 Data Adapters

We implemented three data adapters in the released version of OpenBAN: (1) Xively IoT platform, (2) sMap [21], and (3) SensorAct [11], a research building middleware system. These adapters convert their respective sensor data formats into a collection of $< timestamp, value >$ pairs, represented in Java as a `HashMap <DateTime, Double>`. In addition to receiving sensor data streams from these services, OpenBAN can fetch files from Dropbox that contain ground truth labels or archived sensor data in $< timestamp, value >$ pair format. Both UNIX epoch and ISO8601 *timestamp* format are currently supported. Data adapters for other services, such as Green Button, weather, and variable pricing and grid-load signals can be easily implemented and integrated with OpenBAN.

[7] `https://github.com/nesl/OpenBAN`
[8] `http://www.playframework.org`
[9] `http://quartz-scheduler.org`

4.2 Feature and Model Repository

The released version of OpenBAN contains all the features listed in Table 3. As these features can be directly computed, they are mapped to existing mathematical library functions in Java. OpenBAN also provides a simple user interface that can be used to write a piece of Python code to extract additional feature from sensory data. All the details about each user contributed feature are stored in a JSON file. Similarly, details about available algorithms in each analytical engine and their API endpoints for training and execution are also stored in a JSON file.

4.3 Analytics Engine

OpenBAN leverages OpenCPU[10] as its underlying analytical engines. OpenCPU is an open computing platform which provides RESTful APIs for invoking various library functions in R, a statistical computing language. R provides rich support for a wide variety of machine learning and statistical algorithms. We have implemented an OpenBAN wrapper library for the underlying machine learning algorithms in R, that handles data format issues between OpenBAN and R functions. It has wrapper functions for regression, decision tree, neural network, SVM, naive bayes, and k-NN algorithms. A separate HTTP API endpoint for training and executing each of these algorithm is also included in the model repository. In addition to OpenCPU, we have implemented our own analytics engine called OpenPy[11] for interfacing Python based machine learning packages, such as scikit-learn. OpenPy shares similar goals with OpenCPU and provides RESTful APIs for invoking Python functions.

4.4 User Interface

We implemented a web interface for OpenBAN which allows users to interact with the system components. Using this interface, users (developers and researchers) can create *contextlets* that consists of three intuitive steps: *Aggregate*, *Analyze*, and *Act*. Users are authenticated using their Dropbox credentials through OAuth APIs. Current user interface uses Dropbox as its back-end *Datastore* for storing a user's *App profile*, *Data Repo profile*, and any intermediate data generated during the analysis.

Data adapter instantiation: In order to integrate a sensor data service into OpenBAN, user first creates an instance of a particular *Data adapter*. Such instantiated data adapter is called a sensor *Data Repo*sitory. User needs to provide a name and access credentials such as user name and/or api-key, to instantiate a data adapter. Thereafter, OpenBAN connects to the specified service, pulls a list of available sensor data streams and creates a *Data Repo profile* which is stored in the user's Dropbox account as a JSON file and are referred back whenever OpenBAN needs to read or write the sensor data into the associated service.

Contextlet flow: After instantiating the data adapters, a user can execute the following steps for each phase of the *aggregate-analyze-act* cycle:

Aggregate: In this step, the user aggregates the relevant sensor data by first selecting the training period and a set of relevant sensor data streams (that are then fetched by OpenBAN) from the user specified *Data Repo* profiles. Thereafter, the user specifies the data stream that provides ground truth labels, which may

[10] `http://opencpu.org`
[11] `https://github.com/game-time/OpenPy`

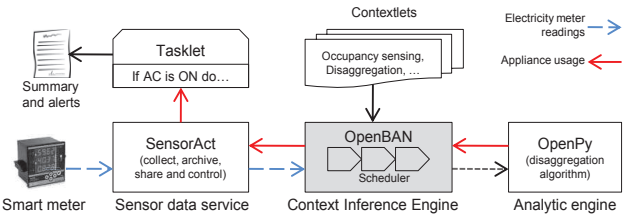

Figure 4: Deployment setup of SensorAct and OpenBAN systems for energy disaggregation application.

possibly be stored under their Dropbox account. Using the integrated visualization tool, the user can also plot the data.

Analyze: After aggregating various sensor data streams the user now needs to 1) select a list of features for each sensor data stream, 2) enter the feature window size in seconds over which the selected features are to be computed, and c) select an analytical algorithm and specify its required parameter(s). Thereafter, the user can initiate the training process by simply clicking on the "Train the model" button. Once a model is trained, the user can schedule its execution over the live sensor data streams by specifying a date range and an execution interval e.g. every 30 seconds between January 7-12, 2014 or every day at 12am. Based on the user specified information, OpenBAN will initiate the execution mode in background, as discussed in Section 3.5.2.

Act: In this final step, the user can select a list of data streams from the *Data Repo* profile to which the executed model output should be communicated.

All the parameters for these three phases are packed into an *App profile*. The User can save an app profile in their Dropbox as a JSON file and reload the parameters into the user interface when required.

5. APPLICATIONS

We developed three energy management applications on top of our prototype implementation of OpenBAN to show the wide applicability of the proposed system. For all these applications, we deployed OpenBAN and analytical engines, OpenCPU and OpenPy, on different virtual machines. Using OpenBAN user interface, we instantiated the required data adapters and then created separate *contextlet* for each application, and deployed them in our test-bed buildings.

5.1 IIIT-Delhi campus testbed

IIIT-Delhi campus was newly constructed two years ago in a space of 25 acres. It consists of five buildings: academic, facilities, faculty apartments (30 houses), mess and hostel (400 dorm rooms) buildings. All these buildings are equipped with a commercial BMS system for managing the various building operations, under the administration of a facility manager (FM). In addition to the commercial BMS system, all the buildings (each floor and flats) were instrumented with over 180 smart meters measuring various electrical parameters. A SensorAct [11] instance, integrated with sMap [21] adapter, was deployed for collecting meter readings and BMS parameters at every 30 seconds.

5.2 Energy disaggregation

Recently, Batra et al. developed NILMTK [23] which contains benchmark NILM algorithms implemented in Python. We leveraged their combinatorial optimization based NILM algorithm to detect refrigerator usage from whole home meter data. A disaggregation function was implemented into the OpenPy analytical engine and it was registered with OpenBAN's Algorithm repository. We chose refrigerator as it contributes significantly to overall energy consumption across different countries [23]. Figure 4 shows the deployment setup of hosting OpenBAN as a private service with SensorAct for this disaggregation task. In this setup, a smart electricity meter installed for a residential apartment was connected with SensorAct using a custom sMAP [21] adapter. Smart meter readings were sampled at 30 seconds interval and archived in SensorAct. We created a *contextlet* in OpenBAN for inferring the current status (on/off status and power consumption) of a refrigerator. This *contextlet* is configured to read the smart electricity meter readings from SensorAct, apply an energy disaggregation algorithm to identify the usage traces of refrigerator, and then push the inferred energy usage status back to SensorAct. Further, the *contextlet* was scheduled to run every 5 minutes and can provide real time refrigerator usage. An alert was setup in SensorAct to notify any unusual refrigerator usage (based on power consumption and duration) to the owner. This experimental application shows that OpenBAN's modular and extensible design makes it easy to integrate very specific building analytics applications such as NILM.

5.3 Sprinkler usage policy violation

In this application, we explore the use of OpenBAN in detecting sprinkler usage compliance with the help of the water usage trace of an entire house, as recorded by the water meter. Home owners can be notified of non-compliant sprinkler usage (sometimes this happens inadvertently when the clock on sprinkler timer goes off due to electricity outage) while the utility company may use it to analyze real-time smart water meter data to detect violators. Since sprinkler systems are mostly automated, they generate a unique pattern when different sprinkler stations are used in sequence. Figure 5 illustrates an example of sprinkler usage pattern along with other water usage activities in the morning.

Data for this experiment has been collected from a single-family home in Los Angeles. Since the water meter in the house was an old mechanical type, we instrumented the main water service pipe feeding the house with a Shenitech STUF-200H ultrasound water flow sensor providing whole-house water meter reading at 1 Hz. A custom software package[12], created for managing diverse forms of sensor data at home, was used to collect the water flow data and upload in real-time to the Xively cloud service. Our testbed household had 9 sprinkler stations scheduled to run at 5 a.m. and 7 p.m.

For this application, we created a *contextlet* in OpenBAN and instantiated the Xively data adapter. Then, we trained a SVM classifier from the Algorithm repository, on 6 months data from June 2013 to November 2013. We chose `mean, standard deviation` and `range` features from OpenBAN's statistical feature repository and `minute of day` using temporal Feature Repository. For all of these features, we chose a time window of 1 min. Since ground truth information is not available, we manually annotated ground truth labels for this training period at 1 min interval. We tested the trained SVM model with water meter readings from the month of December. The model was able to identify the

[12] `https://github.com/nesl/SensorActuatorManager`

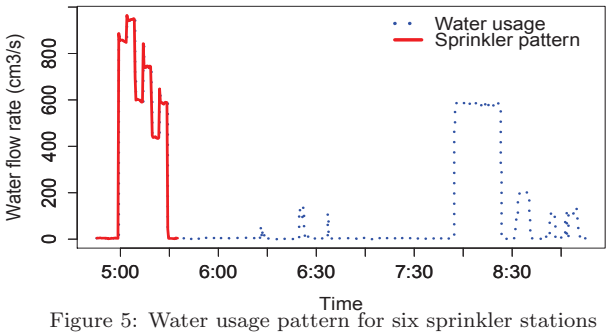

Figure 5: Water usage pattern for six sprinkler stations

sprinkler usage events with 99% accuracy, 96.2% F1-score and 95.2% Matthews Correlation Coefficient. Since water utilities have now begun to install smart water meters that can easily collect water usage remotely in near real-time, an extended version of this experiment can be used to identify customers who are violating the sprinkler usage policy.

5.4 Energy forecasting

In this application, we showcase OpenBAN's capabilities to predict the next hour electricity usage of our test-bed buildings. We created a *contextlet* using OpenBAN's user interface and configured it to use weather (temperature and humidity) and temporal features of the historical energy usage patterns of our test-bed buildings to learn the parameters of a logistic regression classifier instantiated from OpenBAN's Algorithm repository. We selected the `hour-of-day, day-of-week` and `day-of-month` temporal features from OpenBAN's Feature Repository. We chose these features as they represent the energy usage pattern across different periods and seasons. We trained the logistic regression classifier for four weeks of training data and predicted the electricity usage for the next two weeks. With this configuration, OpenBAN was able to predict the hourly usage with 78% accuracy. This *contextlet* was scheduled to run at every hour to provide real time energy forecasting information about our test-bed buildings to the facilities department.

In addition to these three representative applications, we also developed and deployed (1) an indirect occupancy sensing application using the real time Internet traffic, and (2) an anomaly detection application for HVAC systems using the BMS parameters and energy meter readings, in our test-bed buildings.

5.5 System performance

We use the energy forecasting *contextlet* created in the previous section as a candidate application to measure the performance of OpenBAN with in our test-bed building. Particularly, we measure the computation time for executing the *contextlet* and the forecasting model under two difference scenarios: (1) Hosting the analytics engine with in the building, (2) Hosting the analytics engine on the cloud.

We created two *contextlets* for energy forecasting application with similar parameters as described in the previous section. One of them was configured to use local analytical engine hosted on a VM (2×2.5GHz processor, 4 GB RAM) and another was configured to use the public OpenCPU (16×2.8GHz processor, 16 GB RAM) instance[13]. For experimental purpose, these *contextlets* were scheduled to be run at every minute interval. In their each instance of execution,

[13]`https://public.opencpu.org/ocpu/library/`

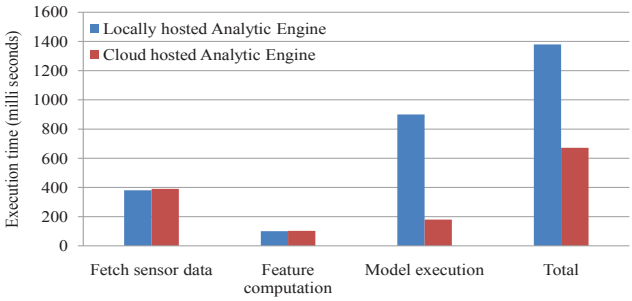

Figure 6: Comparison of execution time between local and cloud hosted analytics engine for the energy forecasting *contextlet*

they read past 1 hour smart meter readings and invoke the forecasting model which was learnt before for one month period. We logged the computation time for 1) fetching past 1 hour meter readings, 2) computing the required features, 3) executing forecasting model (including the network delay), and the total computation time, for each execution instance of the *contextlets*. Figure 6 shows the comparison of these parameters (averaged from 200 execution instances) between local and cloud hosted analytical engines. Although, the total execution times are not directly comparable (because of different hardware configurations), they shows that upper time limit for each *contextlet* instance which is about 700 milliseconds for cloud and 1350 milliseconds for locally hosted anlaytic engines.

6. RELATED WORK

Existing middleware systems related to processing sensor data streams can be broadly classified into four categories: 1) Middlewares for building management, 2) Cloud based Internet of Things (IoT) platforms, 3) Ubiquitous and mobile systems, and 4) General purpose analytical platforms.

Middlewares for building management: In the recent past, several middleware systems have emerged from both research and commercial communities for developing sensory data driven building control applications. BuildingDepot [13] provides a dedicated *ApplicationService* for writing and hosting building control applications. However, no provision has been given within the service to perform common sensory data processing functions. Similarly, SensorAct [11] provides a scripting framework for scheduling periodic and trigger based applications. But its triggers are based only on the arrival of a sensor data event. Further, the APIs provided for reading sensor data are limited to applying only aggregation functions over raw sensor data. In contrast, OpenBAN provides a centralized service for inferring common building contexts from multitude of sensory data and allows applications to subscribe to the desired contextual event and perform actions thereof.

HomeOS [10] presents existing networked devices in homes as PC peripherals. Applications can read the device status or subscribe to events of interest. In either ways, the underlying layers pass raw device data to the applications. On the other hand, BOSS [12] which is specific to commercial buildings, provides a set of operating system services for developing portable and fault-tolerant building control applications. It contains a time series service (TSS) for archiving, querying, and processing sensor data. TSS supports a data transformation language that allows applications to apply a pipeline of numerical operators for data cleaning and trans-

forming the retrieved data. Although these operators are executed within TSS, there is a vertical flow of data between TSS and each application. Our approach is fundamentally different in many ways: (i) applications can read or subscribe to context of their interest instead of invoking data request queries with a combination of numerical operators, (ii) context processing pipelines in OpenBAN can be executed either in real-time or periodically on sensor data streams, and (iii) OpenBAN provides a centralized service for executing context processing pipeline which involves learning and executing complex machine-learning models over multitude of sensor data streams.

Many commercial BMS systems, such as Trane[14], are in widespread use today. They are deployed with a predefined fixed set of applications, e.g. HVAC and fire alarm systems. Their archaic and closed application model makes them difficult to program and extend their functionalities [14]. Whereas, HAS systems, such as Vera, provide a scripting framework for extending the home automation applications. However, their controllers are limited to inferring simple contexts by applying aggregate functions over raw sensor data, e.g., "if garage door is left open then notify the user". Mango automation[15] supports a light weight scripting language for automating building operations but it can be used to identify only simple contexts by fusing multiple sensory data. BuildingOS[16] is an another system specifically designed for energy management operations of a buildings but limited to providing a dashboard for integrating and managing several energy meters.

Cloud based Internet of Things (IoT) platforms: A number of IoT platforms have emerged recently for interconnecting sensors to the Internet and developing novel applications. Examples include Xively, Sen.se[17], among others. While these IoT platforms provide good support for uploading, querying and visualizing the time series sensor data, their inherent support for processing sensory data is limited to applying only simple conditions and aggregation functions. For example Xively supports threshold-based triggers by applying simple relational operators over live sensory data streams. The application programming framework in Sen.se allows developers to write *Data funnels* which fuse multiple sensor data streams based on aggregation functions.

Nimbits takes a different approach by supporting execution of several data filters on streaming sensor data (e.g. to detect faulty data) and by providing a loose integration with Wolfram Alpha[18]. However, such loose coupling prevents the use of Wolfram Alpha for continual real-time analytics and offers limited support for statistical machine learning capabilities. IFTTT[19] provides support for creating home automation recipes using WeMo[20] devices. Its "if-then" model triggers are limited to inferring contexts which are directly derivable from individual raw sensor data. MathEngine in SensorCloud allows developers to write sophisticated analytics scripts to process live sensor data streams on the cloud.

Mobile and Ubiquitous computing platforms: Systems for monitoring rich context information from sensor data, proposed for domains such as mobile computing, are the closest in spirit to OpenBAN (e.g., MobiCon [24] and Darwin phones [25]). OpenBAN shares similar goals with Auditeur [26] for creating a context monitoring pipeline which involves a learning phase and an execution phase of previously learned models. While the approach for inferring a context from sensor data streams is similar in both buildings and mobile systems, OpenBAN provides additional support for extending the system. It allows developers to add and reuse, features and classification algorithms to its repository.

Earlier works on "context-aware" applications for ubiquitous environments are different from context monitoring approaches in buildings. While the former focuses on reasoning about the environment from a knowledge repository, the latter focuses on monitoring the context from a multitude of sensory data events. Several researchers have conducted work [5,19] on modeling, analyzing and inferring rich context information in buildings. However, all of them have pursued a narrow goal of coupling the sensors with a particular building control application, such as occupancy based HVAC control. Unlike them, OpenBAN seeks to create an ecosystem for deriving inferences in which diverse building related applications can subscribe to contexts of their interest computed over streaming and historical sensor data.

General purpose analytical platforms: Standalone tools such as Weka[21] and cloud-based general purpose analytics platforms such as BigML[22] and OpenML[23] have attempted to make complex machine learning algorithms accessible to the users. For example, Weka provides an interface for users to create data flows for common machine learning algorithms. Other cloud based systems provide web APIs and user interfaces for experimenting with complex machine learning algorithms. But these platforms and services are primarily designed as batch processing engines as opposed to one that would perform real-time continual operation on time series sensor data. Further, all these services require features extracted from raw sensory data as their input. Esper[24] is an in-memory Complex Event Processing (CEP) engine which provides query based aggregation and data fusion methods. But it is limited to applying only aggregation functions over different time windows.

7. FUTURE WORK

There are many enhancements that we have planned to include in the proposed system. At present, OpenBAN receives sensor data streams from existing sensor data services. We assume that such existing systems already take care of data cleaning issues. However, it is likely that real word sensors in buildings misbehave and may send faulty data. So we have planned to include a separate data pre-processing component to the system. In the current implementation, OpenBAN adapters are only pull-based, adapter reads the data from sensor data services. We plan to implement push-based adapters, where the adapter listens to sensory events. While such a push based model would be suitable for processing streaming data, adapters need to provide additional buffering, as most of the learning based algorithms work on features extracted over a time window.

In the current implementation, user needs to switch the contextlet from training and execution mode manually. An-

[14] http://www.trane.com
[15] http://infiniteautomation.com
[16] http://buildingos.com
[17] http://open.sen.se
[18] http://www.wolframalpha.com
[19] http://ifttt.com
[20] http://www.belkin.com/us/wemo
[21] http://www.cs.waikato.ac.nz/ml/weka/
[22] http://bigml.com
[23] http://openml.org
[24] http://esper.codehaus.org

other important feature would be making an adaptive analytics engine that will automatically switch from learning mode to execution mode on the basis of the accuracy of the trained model by applying cross validation methods. Further, in order to quantitatively evaluate the generality and usability of OpenBAN, we intended to conduct a survey among a list of developers after asking them to design the same representative applications that we have developed.

8. CONCLUSION

In this paper, we proposed OpenBAN, an open sensor data analytics middleware which facilitates the development of novel building energy management applications. We described the architecture of OpenBAN that consists of (a) Data adapters to integrate multiple sensor data repositories into the system; (b) Feature Repository that provides commonly used features that are derived from various sensor data streams; (c) Model Repository that contains commonly used analytics algorithms and their trained instances to infer various context information in buildings; (d) Context Inference Engine for coordinating other components and scheduling the execution of an analytics application flow; and (e) Analytics engines for executing the machine learning algorithms. An implementation of OpenBAN, with these various components has been released as open source. Using OpenBAN user interface, users can create the *Aggregate-Analyze-Act* flow for an analytics application for modern smart buildings. We developed several real-world energy management applications to show the wide utility of OpenBAN . We also perform a preliminary system performance evaluation and find it to be satisfactory for the intended application.

9. ACKNOWLEDGMENTS

This material is based upon work partially supported by the US NSF under award # CNS-1143667 and CNS-0905580, by IARPA, USA, and by DEITy, India under grant number DeitY/R&D/ITEA/4(2)/2012. Authors will also like to acknowledge the support provided by ITRA project, funded by DEITy, Government of India, under grant with Ref. No. ITRA/15(57)/Mobile/HumanSense/01. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

10. REFERENCES

[1] GeSI, *SMART 2020: Enabling the low carbon economy in the information age.* Climate Group, 2008.

[2] A. Research, "1.5 million home automation systems installed in the us this year," Nov 2012 (accessed July 25, 2013).

[3] U. E. I. Administration, "How many smart meters are installed in the u.s. and who has them?," 2013 (accessed July 25, 2013).

[4] S. Afshari, S. Mishra, J. Wen, and R. Karlicek, "An adaptive smart lighting system," in *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pp. 201–202, ACM, 2012.

[5] J. Lu, T. Sookoor, V. Srinivasan, G. Gao, B. Holben, J. Stankovic, E. Field, and K. Whitehouse, "The smart thermostat: using occupancy sensors to save energy in homes," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pp. 211–224, ACM, 2010.

[6] V. Erickson, M. Carreira-Perpinan, and A. Cerpa, "Observe: Occupancy-based system for efficient reduction of hvac energy," in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pp. 258–269, IEEE, 2011.

[7] T. Weng, B. Balaji, S. Dutta, R. Gupta, and Y. Agarwal, "Managing plug-loads for demand response within buildings," in *Proceedings of the Third ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pp. 13–18, ACM, 2011.

[8] G. W. Hart, "Nonintrusive appliance load monitoring," *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, 1992.

[9] C. Chen, D. J. Cook, and A. S. Crandall, "The user side of sustainability: Modeling behavior and energy usage in the home," *Pervasive and Mobile Computing*, 2012.

[10] C. Dixon, R. Agarwal, A. Saroiu, and P. Bahl, "An operating system for the home," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, 2012.

[11] P. Arjunan, N. Batra, H. Choi, A. Singh, P. Singh, and M. B. Srivastava, "Sensoract: a privacy and security aware federated middleware for building management," in *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pp. 80–87, ACM, 2012.

[12] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler, "Boss: building operating system services," in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.

[13] 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*, pp. 1–8, ACM, 2013.

[14] A. Krioukov, G. Fierro, N. Kitaev, and D. Culler, "Building application stack (bas)," in *Proceedings of the Fourth ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pp. 72–79, ACM, 2012.

[15] R. P. Singh, S. Keshav, and T. Brecht, "A cloud-based consumer-centric architecture for energy data analytics," in *Proceedings of the fourth international conference on Future energy systems*, pp. 63–74, ACM, 2013.

[16] S. Darby, "The effectiveness of feedback on energy consumption," *A Review for DEFRA of the Literature on Metering, Billing and direct Displays*, vol. 486, 2006.

[17] L. A. D. of Water and Power, "New watering schedule," 2013 (accessed July 25, 2013).

[18] AEO, "US Energy Information Administration," *AEO2011: Annual Energy Outlook*, April 2011.

[19] C. Beckel, L. Sadamori, and S. Santini, "Automatic socio-economic classification of households using electricity consumption data," in *Proceedings of the fourth international conference on Future energy systems*, pp. 75–86, ACM, 2013.

[20] W. Kleiminger, C. Beckel, T. Staake, and S. Santini, "Occupancy detection from electricity consumption data," in *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, pp. 1–8, ACM, 2013.

[21] 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*, pp. 197–210, ACM, 2010.

[22] Y. Agarwal, R. Gupta, D. Komaki, and T. Weng, "BuildingDepot: An Extensible and Distributed Architecture for Building Data Storage, Access and Sharing," in *Fourth ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings*, BuildSys 2012, 2012.

[23] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh, and M. Srivastava, "Nilmtk: An open source toolkit for non-intrusive load monitoring," in *Proceedings of the 5th international conference on Future energy systems*, pp. 265–276, ACM, 2014.

[24] Y. Lee, S. Iyengar, C. Min, Y. Ju, S. Kang, T. Park, J. Lee, Y. Rhee, and J. Song, "Mobicon: a mobile context-monitoring platform," *Communications of the ACM*, vol. 55, no. 3, pp. 54–65, 2012.

[25] E. Miluzzo, C. T. Cornelius, A. Ramaswamy, T. Choudhury, Z. Liu, and A. T. Campbell, "Darwin phones: the evolution of sensing and inference on mobile phones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 5–20, ACM, 2010.

[26] S. Nirjon, R. F. Dickerson, P. Asare, Q. Li, D. Hong, J. A. Stankovic, P. Hu, G. Shen, and X. Jiang, "Auditeur: A mobile-cloud service platform for acoustic event detection on smartphones," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pp. 403–416, ACM, 2013.